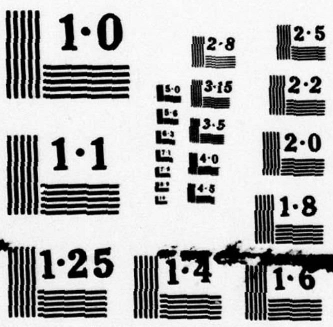
AD-A067 690   SYRACUSE UNIV N Y   F/G 9/2
BAYESIAN SOFTWARE PREDICTION MODELS. VOLUME V. SUMMARY OF TECHN--ETC(U)
OCT 78 A L GOEL   F30602-76-C-0097
UNCLASSIFIED   TR-78-5   RADC-TR-78-155-VOL-5   NL

1 OF 2
ADA
067690


1·0
2·8
2·5
3·15
2·2
3·5
1·1
2·0
4·0
4·5
1·8
1·25
1·4
1·6

RADC-TR-78-155, Volume V (of five)
Final Technical Report
October 1978

BAYESIAN SOFTWARE PREDICTION MODELS
Summary of Technical Progress

Amrit L. Goel

Syracuse University

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

RADC-TR-78-155, Volume V (of five) has been reviewed and is approved for publication.

APPROVED: Alan N. Sukert
ALAN N. SUKERT
Project Engineer

APPROVED: [illegible]
WENDALL C. BAUMAN, Colonel, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss
JOHN P. HUSS
Acting Chief, Plans Office

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19 ~~REPORT DOCUMENTATION PAGE~~	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 18 RADC-TR-78-155, Vol V ~~(of five)~~ - 5 / 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER 9
4. TITLE ~~(and Subtitle)~~ 6 BAYESIAN SOFTWARE PREDICTION MODELS. Volume V. Summary of Technical Progress.	5. TYPE OF REPORT & PERIOD COVERED Final Technical Report, Jan 76 - Apr 78,
	6. PERFORMING ORG. REPORT NUMBER 14 TR- Technical Report No. 78-5
7. AUTHOR(s) 10 Amrit L. Goel	8. CONTRACT OR GRANT NUMBER(s) 15 F30602-76-C-0097 ✓
9. PERFORMING ORGANIZATION NAME AND ADDRESS Syracuse University Syracuse NY 13210	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62702F 16 5581 17 14
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISIS) Griffiss AFB NY 13441	12. REPORT DATE 11 Oct ~~ober~~ 1978
	13. NUMBER OF PAGES 94
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same 12 102p.	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.	
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Same	
18. SUPPLEMENTARY NOTES RADC Project Engineer: Alan N. Sukert (ISIS)	

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software Error Prediction	Software Demonstration Testing
Imperfect Debugging	Software Error Correction
Imperfect Maintenance	Bayesian Models
Software Modeling	Bayesian Inference

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report summarizes the technical activities pursued under Contract F30602-76-C-0097, Bayesian Software Prediction Models, with Rome Air Development Center. Research work, discussed in previous volumes, discussing imperfect debugging and imperfect maintenance software performance models, is summarized and some additional work in development of software reliability demonstration test plans is described.

DD FORM 1 JAN 73 1473 339600 UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EVALUATION

The necessity for more complex software systems in such areas as command and control and intelligence has led to the desire for better methods for predicting software errors and reliability to insure that software produced is of higher quality and of lower cost. This desire has been expressed in numerous industry and Government sponsored conferences, as well as in documents such as the Joint Commanders' Software Reliability Working Group Report (November 1975). As a result, numerous efforts have been initiated to develop and validate mathematical models for predicting such quantities as the number of remaining errors in a software package and the time to achieve a desired level of reliability. In addition, efforts have been initiated to develop better methods for determining when a software package should be released to a potential user. However, these efforts have not produced measures with the desired accuracy or confidence for general applicability.

This effort was initiated in response to this need for developing better and more accurate software error prediction and demonstration tests and fits into the goals of RADC TPO No. 5, Software Cost Reduction in the subthrust of Software Quality (Software Modeling). This report summarizes the development of mathematical models for predicting quantities such as the expected number of errors during both software development and software maintenance. These models assume errors are not corrected with probability 1, i.e. imperfect development and maintenance. The report also describes the development of statistical tests for determining whether a software package should be accepted or rejected after completion of testing. The importance of these developments is that they represent the first attempt to develop both software error prediction models that incorporate imperfect debugging and thus more closely reflect the actual software error detection and correction process, and software demonstration tests that allow better statistical criteria for accepting a software package.

The theory and equations developed under this effort will lead to much needed predictive measures for use by software managers in more accurately tracking software development projects in terms of stated error and reliability objectives. In addition, the associated confidence limits and other related statistical quantities developed under this effort will insure more widespread use of these techniques. The acceptance criteria developed will permit better control of the release of software packages so that software is not given to a potential user before it is ready for operational usage. Finally, the measures developed under this effort will be applicable to current software development projects and thus help to produce the high quality, low cost software needed for today's systems.

[illegible]
ALAN N. SUKERT
Project Engineer

ABSTRACT

This report provides a summary of the technical activities pursued under Contract F30602-76-C-0097 with RADC during January 1976-April 1978. Research work discussed in previous reports under this contract is summarized and some additional work is described. Also included is a brief description of research in progress.

1. INTRODUCTION

During the past ten years the field of software engineering has grown considerably in importance and scope. A primary motivation for this growth has come from an ever increasing cost of developing and maintaining software systems. This is specially true for the DOD which needs high quality, low cost software for its operations. As a result of this increased importance, various fields within software engineering are maturing into disciplines of study and research, for example, software design techniques, structured programming and other improved programming methodologies, program testing and debugging techniques, software performance modelling, and techniques of program validation and verification. The ultimate objective of studies in all these fields is to develop tools that will be useful in the design, development and operational phases of the software life cycle. The objective of studies dealing with software error analysis and modelling has been to develop analytical tools which can be used for improving software performance. Such studies can be classified into one (or both) of two categories. In the first category the emphasis is on the analysis of software error data collected from small or large projects, during development and/or operational phases. Studies in the second category are primarily aimed at the development of analytical models which are then used to obtain the reliability and other quantitative measures of software performance.

Typical of the first category are the studies by Akiyama [1] Belady and Lehman [3], Fries [6], Endres [5], Baker [2], Motley et al [18], Miyamoto [16], Willman et al [35], Schneidewind [26], Shooman et al [29], Sukert [30,31], Rye et al [24], Thayer et al [32], and Wagoner [34]. These studies range in size from an analysis of small data sets (108 errors), e.g. Wagoner [34], to analysis of large sets (3500 errors), e.g. Thayer et al [32] and encompass data from an on-line system [16], an operating system [3], to that from the Apollo project [24].

In the second category of papers, several models have been proposed and studied during the last six years. These include 'exponential type' models of Shooman [28], Jelinski and Moranda [11,12], and Schick and Wolverton [25]; models based on the non-homogeneous Poisson process proposed by Goel and Okumoto [9] and Schneidewind [27], and a Bayesian model by Littlewood and Verrall [15]. Halstead [10] has developed a theory based on 'software physics' for various measures of the performance of a software system. Musa [19] has introduced a model which is based on a large number of parameters derived from the software system being modelled. Trivedi and Shooman [33] consider a Markov model in which they incorporate the time spent for removal of errors.

Most of the above studies assume that errors are removed with certainty when detected. The purpose of the investigation summarized in this report is to develop and study models for software performance which account for the probabilistic nature of the programmer's action during debugging and operational phases of the software system, to provide a methodology for classical and Bayesian inference for various quantitative measures of performance, to develop optimum Bayesian software correctional limit policies, and to develop software reliability demonstration test plans. Results of these studies are summarized in Sections 2 through 6.

2. A SOFTWARE PERFORMANCE MODEL UNDER IMPERFECT DEBUGGING

The purpose of this modelling effort was to establish quantitative measures for software systems by incorporating the probabilistic nature of the programmer's actions during the debugging phase. Such occurrences have been termed as recurrent errors by Fries [6], Thayer [32] and Willman et al [35], erroneous debugging by Miyamoto [16], bad fixes by Jones [13] and accounted as an error reduction factor by Musa [19]. In this study we call them the imperfect debugging errors.

Even though the presence of the imperfect debugging phenomenon has been known, no published model, with the possible exception of Musa's error reduction factor, provides an explicit way to account for it in software performance prediction. The model and other related results for this topic are summarized below. Details of this work are given in [7]. These results are useful for software development personnel in establishing manpower requirements to achieve a desired quality in the system before it is released for operational use. Trade-off studies between cost of debugging and software quality can be undertaken using the results given below.

2.1 Model and Main Results

The following parameters are used for model development and analysis:

N = the initial number of errors in the software system at the beginning of the debugging activity

p = probability that the error causing a software failure is removed when detected

$q = 1-p$, the probability of imperfect debugging

λ = software error occurrence rate

Let a random variable $X(t)$ denote the number of errors remaining in the system at time t. Then $X(t)$ describes the state of the system at time t. We consider the stochastic process $\{X(t), t \geq 0\}$ to be a semi-Markov process with the one-step transition probability, $Q_{ij}(t)$, the probability that the next failure resulting in j remaining errors will be by time t when a software system has i remaining errors at time zero. Then

$$Q_{ij}(t) = \begin{cases} p(1-e^{-i\lambda t}) & \text{if } j=i-1 \\ q(1-e^{-i\lambda t}) & \text{if } j=i \end{cases}$$

If we start with N errors at $t=0$, we are interested in the expressions for various quantities that describe the software system performance. These quantities are given below.

2.1.1 <u>Distribution to time to a specified number of remaining errors</u>

Let $G_{N,n_0}(t) \equiv P(T_{N,n_0} \leq t)$ be the cdf of the time T_{N,n_0}, required to obtain a software system with n_0 remaining errors, $n_0=0,1,2,\ldots,N-1$. Then

$$G_{N,n_0}(t) = \sum_{j=1}^{N-n_0} B_{N,j,n_0}\{1-e^{-(n_0+j)p\lambda t}\},$$

where

$$B_{N,j,n_0} = \frac{N!}{n_0!\,j!\,(N-n_0-j)!}(-1)^{j-1} \cdot \frac{j}{n_0+j}.$$

The expected time required to obtain a software system with n_0 remaining errors is given by

$$E[T_{N,n_0}] = \sum_{j=1}^{N-n_0} B_{N,j;n_0}/(n_0+j)p\lambda .$$

2.1.2 <u>Probability distribution of a given number of remaining errors at time t</u>

Probability that there are n_0 remaining errors at time t is

$$P_{N,n_0}(t) = P\{X(t)=n_0|X(0)=N\}$$
$$= G_{N,n_0}(t)-G_{N,n_0-1}(t) , \quad n_0=0,1,2,\ldots,N ,$$

where

$$G_{N,N}(t)=1$$

and

$$G_{N,-1}(t) = 0 .$$

Also, the expected number of errors remaining at time t is

$$E[X(t)|X(0)=N] = Ne^{-p\lambda t} .$$

2.1.3 <u>Expected number of total and imperfect debugging errors</u>

The expected total number of errors, $M_N(t)$, and errors due to imperfect debugging, $D_N(t)$, during a debugging time period t are given by

$$M_N(t) = \frac{N}{p}(1-e^{-\lambda pt}) ,$$

and

$$D_N(t) = q \cdot M_N(t) .$$

2.1.4 Reliability function

The software system reliability between (k-1)st and kth failures is given by

$$R_k(x) = \sum_{j=0}^{k-1} \binom{k-1}{j} p^{k-j-1} q^j e^{-\{N-(k-j-1)\}\lambda x} .$$

2.1.5 Gamma approximation

The computation of the quantity B_{N,j,n_0} and hence $G_{N,n_0}(t)$ becomes cumbersome when N is large. We have found that the following gamma approximation yields satisfactory results for large software systems with large values of N .

$$G_{N,n_0}(t) \approx \int_0^t \frac{\beta \cdot (\beta x)^{\alpha-1}}{\Gamma(\alpha)} \cdot e^{-\beta x} dx ,$$

where the scale parameter β and the shape parameter α are estimated as

$$\beta = p\lambda \frac{\sum_{j=n_0+1}^{N} 1/j}{\sum_{j=n_0+1}^{N} 1/j^2} ,$$

and

$$\alpha = \frac{\{\sum_{j=n_0+1}^{N} 1/j\}^2}{\sum_{j=n_0+1}^{N} 1/j^2} .$$

2.1.6 Numerical examples

To illustrate the usefulness of the above results consider a software system with $N=100$, $\lambda=0.02$. The probability distributions of times to $n_0=0(1)10$ remaining errors are obtained as above and are shown in Figure 2.1 for $p=0.9$. We see that at $t=200$, say, the probability of having zero errors in the system is approximately 0.1, of one error about 0.15, of 2 about 0.23 and so on. Plots of expected number of remaining errors at various times are given in Figure 2.2 for $p=0.8(0.05)1.00$. For $p=0.9$ and $t=100$, there will be about 18 errors left in the system. From a study of these plots, one can plan the resource requirements and also conduct trade-off studies between available resources and resulting product.

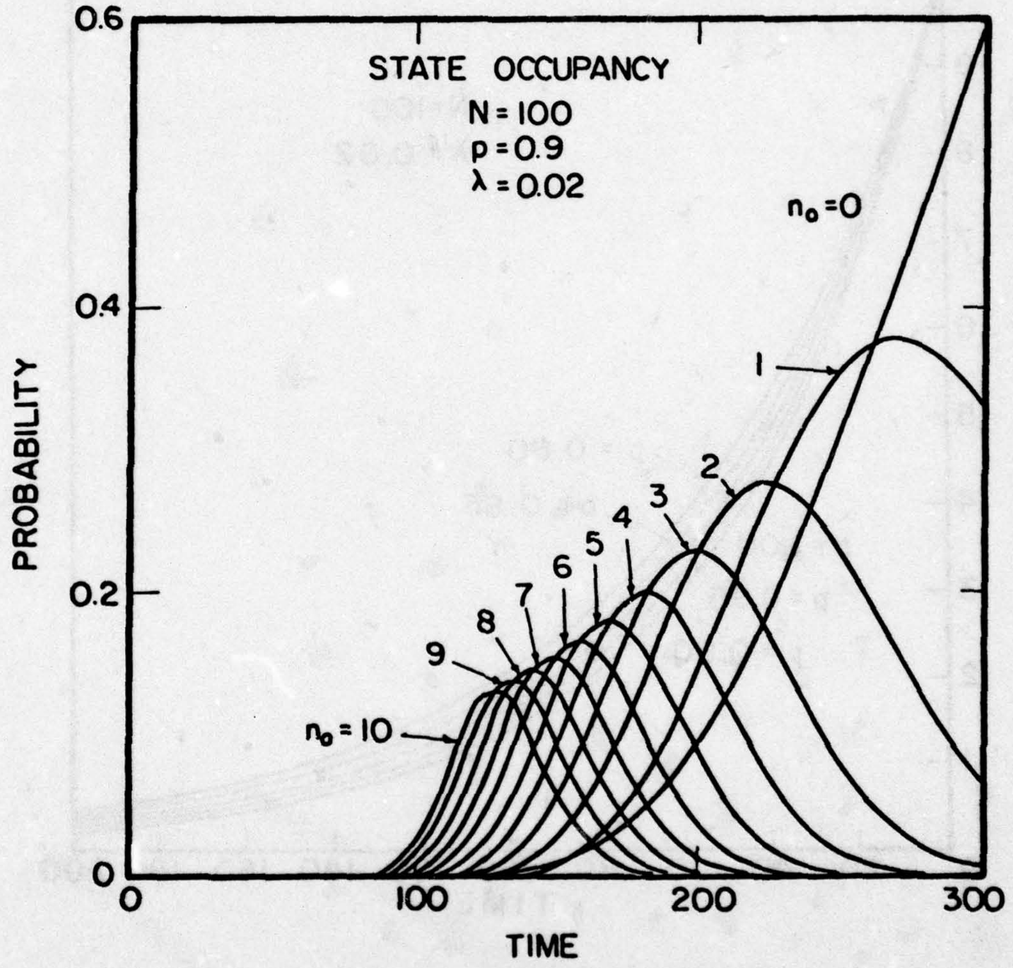


Figure 2.1 Probability Distribution of Time to n_0 Remaining Errors

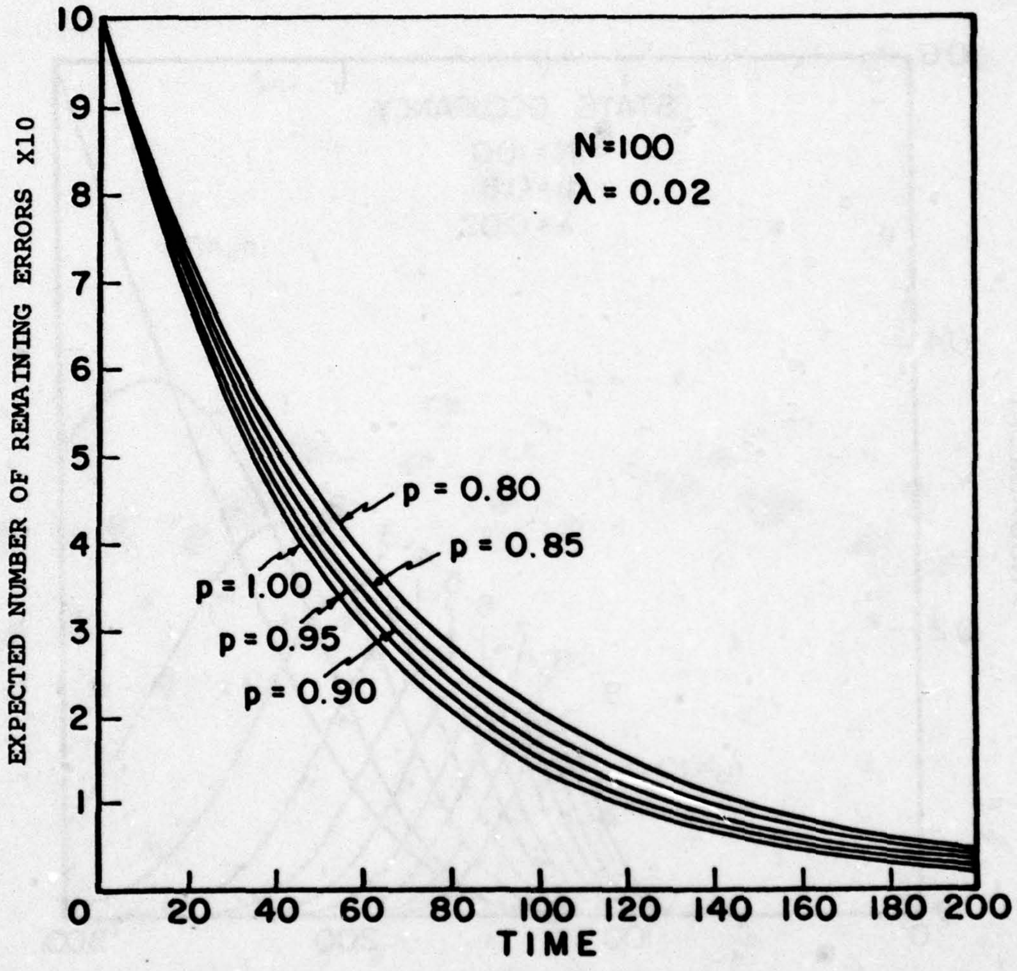


Figure 2.2 Expected Number of Remaining Errors versus Time

2.2 Analysis of Total and Imperfect Debugging Errors in a Real-Time Control System

The results described in Section 2.1 were used to analyze the software error data from a large-scale software project - a real-time control system for a land-based radar system developed by Raytheon Co. in a modular fashion, see Willman [35]. The data base was extracted from 2165 Software Problem Reports written against 109 operational software modules over the development phase. Details of this analysis are given in Appendix A.

3. AVAILABILITY ANALYSIS OF SOFTWARE SYSTEMS UNDER IMPERFECT MAINTENANCE

In this section we describe a model develope for the operational phase of the software system subject to imperfect error maintenance and also incorporate the time spent for error maintenance.

3.1 Model and Performance Measures

The following parameters are used in this model:

N = Initial number of errors in the software system at the beginning of software operation

p = Probability that the error causing a software failure is removed/maintained when detected.

$q = 1-p$, the Probability of imperfect maintenance/removal

λ_i = Software error occurrence rate per unit time when there are i errors in the system

μ_i = Software error correction/maintenance rate per unit time when i errors remain in the system

Consider a stochastic process $\{X(t),\ t\geq 0\}$, whose states are defined as

$$X(t) = \begin{cases} i & \text{if the software system is operational while there are } i \text{ errors in the system } (i=0,1,2,\ldots,N) \\ D & \text{if the software system is down for error removal i.e., for maintenance.} \end{cases}$$

Then, the process $\{X(t),\ t\geq 0\}$ forms a semi-Markov process with the one step transition probability (the probability that the next up-down cycle, resulting in j remaining errors, will be completed by time t when a software package has i remaining errors at time zero) given by

$$Q_{ij}^{(D)}(t) = \begin{cases} p(1-e^{-\lambda_i t})\cdot(1-e^{-\mu_i t}) & \text{if } j=i-1 \\ q(1-e^{-\lambda_i t})\cdot(1-e^{-\mu_i t}) & \text{if } j=i \\ 0 & \text{otherwise} \end{cases}$$

Based upon the above model, expressions for the following performance measures of the software system are derived by Okumoto and Goel [21]:

(i) Distribution of time from N to a specified number of remaining errors n_0

(ii) Expected time required for the system to go from N to n_0 errors, $E[T_{N,n_0}]$

(iii) Probability of the system being operational at some time with n_0 remaining errors, $P_{N,n_0}(t)$.

(iv) Software system availability, i.e., the probability of the system being operational at time t, $A(t) = \sum_{n_0=0}^{N} P_{N,n_0}(t)$

(v) Probability that the number of errors remaining in the system is n, $P[\overline{N}(t)=n]$ and the expected number of errors at t, $E[\overline{N}(t)]$

(vi) Expected number of errors detected and corrected by t, denoted by $M_N^D(t)$ and $M_N^C(t)$, respectively.

The above quantities are useful for software managers for estimating the time and manpower requirements to achieve a desired level of performance during the operational phase of the software system.

3.2 Numerical Example

For illustration purposes consider the case when $\lambda_i = i\lambda$, $\mu_i = i\mu$, N=100, p=0.9, $\lambda=0.02$ and $\mu=0.05$. Using the expressions for various performance measures from [21], the plots of state occupancy probabilities and availability are given in Figure 3.1 and the plots of $M_N^D(t)$ and $M_N^C(t)$ are given in Figure 3.2. From a study of such plots for various values of λ, μ and p, one can obtain adequate information about the behavior of the software system as well as about the resource requirements to achieve a desired level of performance. Thus, if p is known to be 0.9, and the values of λ and μ that can be provided for, then the system availability at t=300 will be approximately 0.5. If this is not satisfactory, one has to provide additional or better resources that will yield better values for one or more parameters.

3.3 A Nomogram for the Expected Time to a Specified Number of Errors and to Determine Manpower Requirements

We present a simple nomogram to calculate the expected time required to remove a specified number of errors from a software system which will satisfy the desired performance requirements. We consider the case when $\lambda_i = i\lambda$ and $\mu_i = i\mu$, i.e., when the error detection and error correction rates are proportional to the number of remaining errors with λ and μ, respectively, the constants of proportionality. Letting the ratio $\lambda/\mu = \rho$, we have

$$E[T_{N,n_0}] = \frac{1+\rho}{\rho\lambda} \sum_{i=n_0+1}^{N} \frac{1}{i} .$$

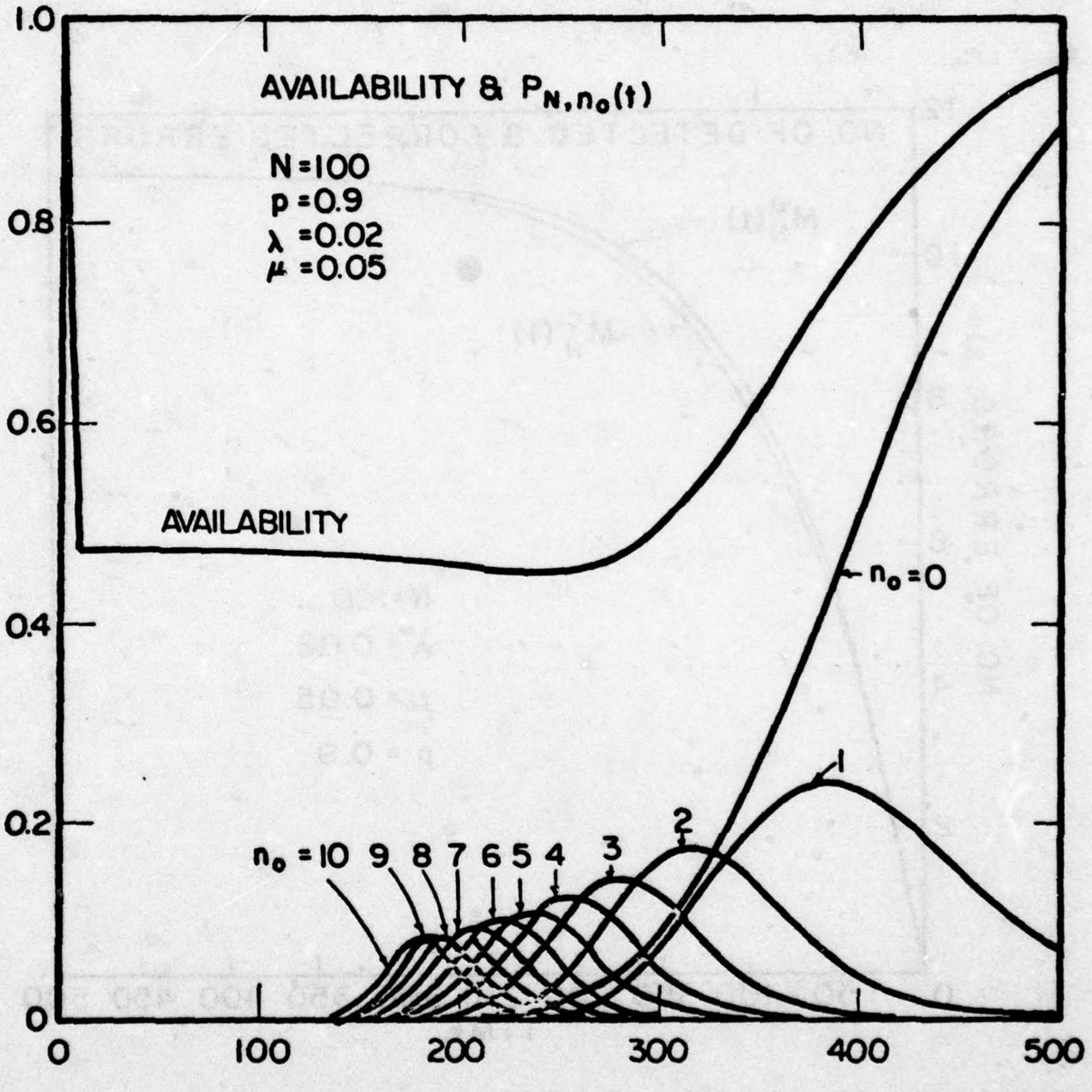


Figure 3.1 Plots of State Occupancy Probabilities and Software System Availability

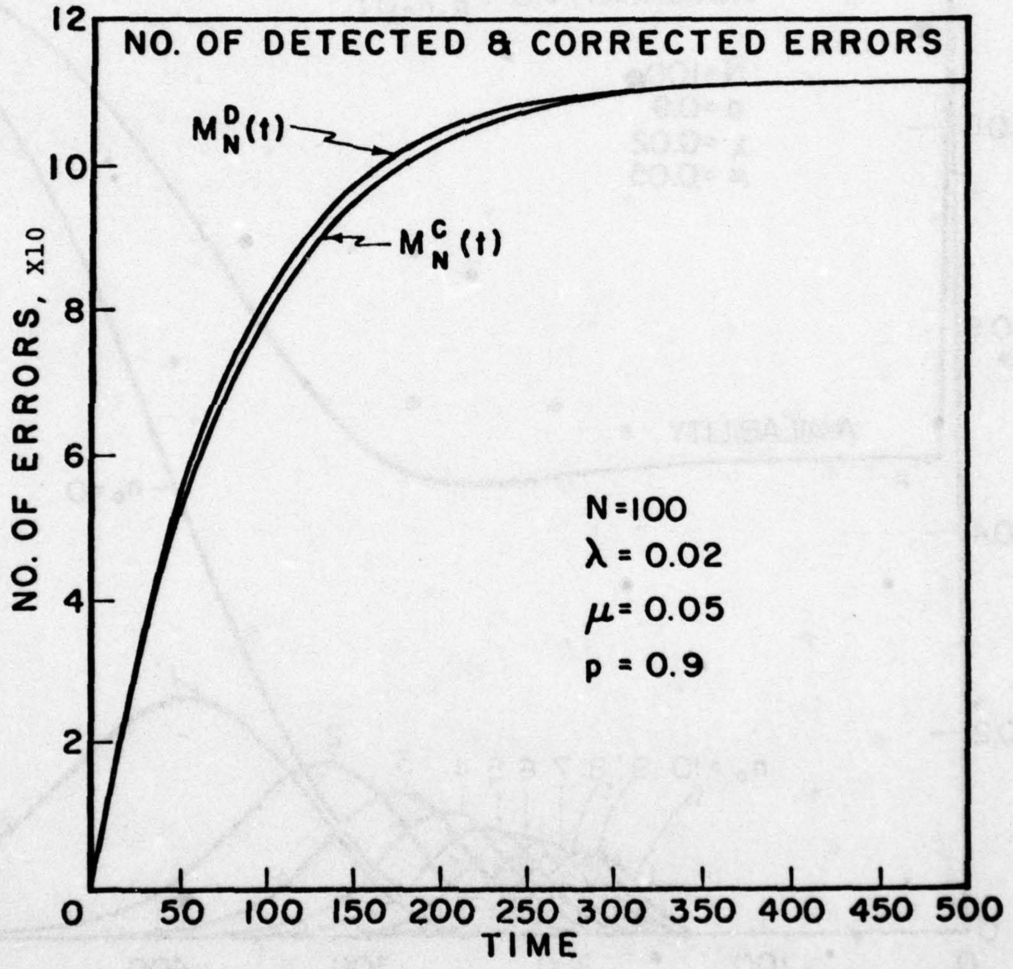


Figure 3.2 Expected Number of Software Errors Detected and Corrected by Time t

For a large software system, this can be approximated as

$$E[T_{N,n_0}] \approx \frac{(1+\rho)}{p\lambda} \log(\frac{N+1}{n_0+1}) .$$

Letting $\Phi(\phi,\rho)=(1+\rho)\phi$, where $\phi=\log(\frac{N+1}{n_0+1})$, we get

$$E[T_{N,n_0}] = \frac{\Phi(\phi,\rho)}{p\lambda} .$$

A nomogram which gives the contours of $\Phi(\phi,\rho)$ in the (ϕ,ρ) phase is given in Figure 3.3. To illustrate the use of this nomogram, consider the case when $N=100$, $p=0.9$, $\lambda=0.02$ per day, $\mu=0.05$ per day and the desired value of n_0 is 10. We proceed as follows.

Step 1. Compute $\rho = \frac{\lambda}{\mu} = \frac{0.02}{0.05} = 0.4$ and $\phi=\log(\frac{N+1}{n_0+1})=\log(\frac{100+1}{10+1})=0.963.$

Step 2. Corresponding to $\phi=.963$ and $\rho=0.4$, from Figure 3.3 the value of $\Phi(.963,0.4)$ is 3.2.

Step 3. Compute the value $E[T_{100,10}] = \frac{\Phi(\phi,\rho)}{p\lambda} = \frac{3.2}{(0.9)(0.02)} = 177.8$

Thus, for the given conditions the expected time required to go from 100 to 10 errors is 177.8 days.

Now we consider another application of this nomogram to show how it can be used in determining the manpower requirements for a specified objective of remaining errors. Suppose we want to go from $N=1000$ to $n_0=10$ errors in $E[T_{1000,10}]=500$ days when the error occurrence rate is $\lambda=0.01$ errors per day and the probability p of perfect correction is 0.95. We are interested in determining the manpower requirement to accomplish this objective.

The first thing to determine is the value of μ that will satisfy these requirements. We know that

$\Phi(\phi, \rho) = 2.5(0.5)25$

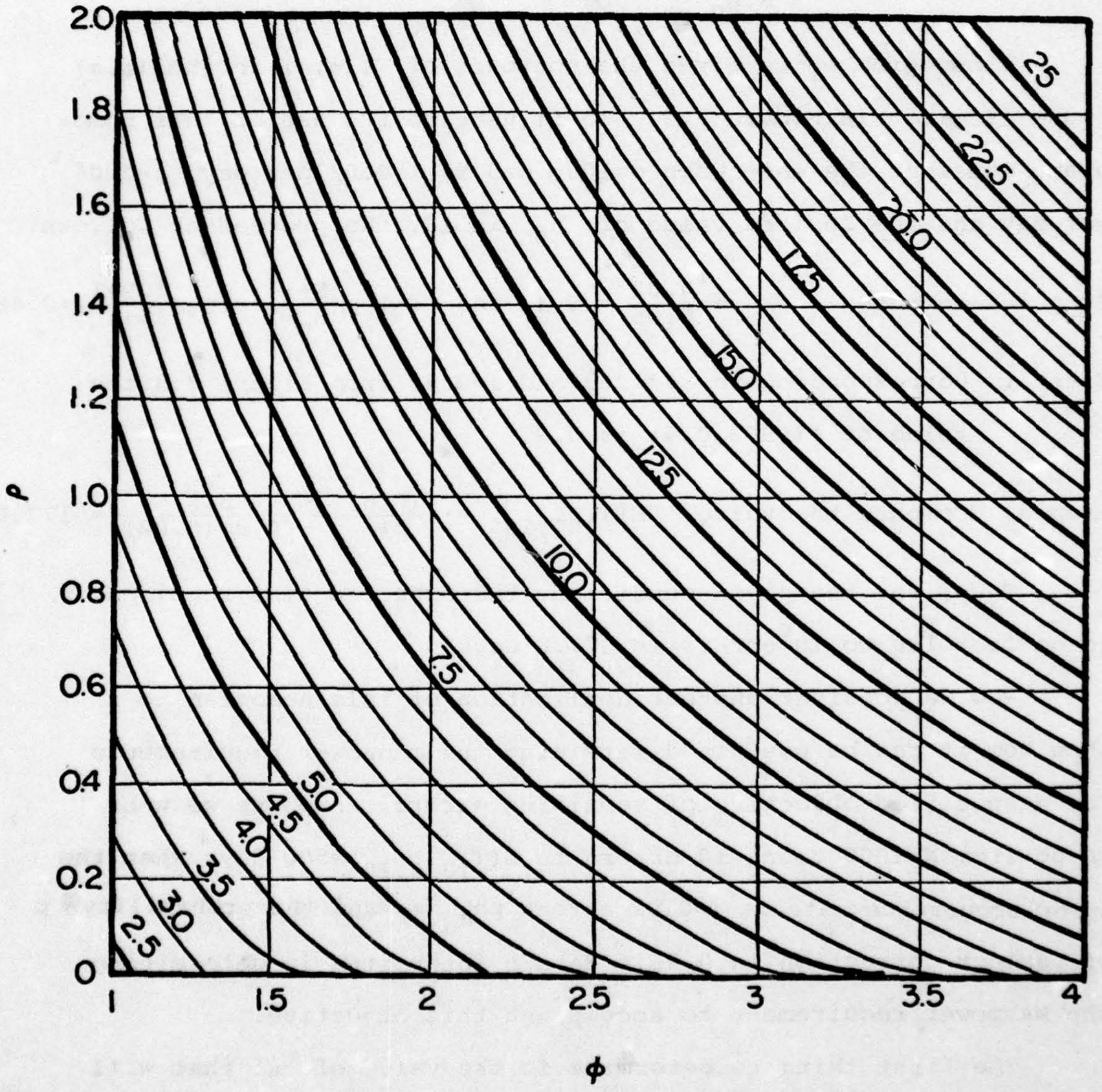


Figure 3.3 A Nomogram for the Expected Time to a Specified Number of Errors

$$\Phi(\phi,\rho) = p\lambda E[T_{N,n_0}]$$

and hence

$$\Phi(\phi,\rho) = (.95)(.01)(750) = 7.125.$$

From the nomogram in Figure 3.3, for $\Phi(\phi,\rho)=7.125$ and $\phi=\log(\frac{1001}{11})=1.96$, the value of $\rho=0.58$ and hence $\mu=\lambda/\rho=0.0172$ errors/day. If the error removal rate per person per day is 0.01, then we will need 1.7 people to meet the desired objective.

4. BAYESIAN AND CLASSICAL INFERENCE FOR THE IMPERFECT DEBUGGING AND MAINTENANCE MODELS

In this section we describe two methods for statistical inference of the parameters of the models described in Sections 2 and 3. The first one is the classical approach based on maximum likelihood estimation and the second is a Bayesian approach based on the prior distributions of the unknown parameters. The parameters under consideration are the initial number of software errors N, the error occurrence rate for each error λ, and the probability of perfect debugging p. An additional parameter for the imperfect maintenance model is μ. We give only the method for the model of Section 2. The same procedure can be used for the model of Section 3.

The available data for estimation purposes is generally given as $\underline{t}=(t_1,t_2,\ldots,t_n)$, the times between software failures and $\underline{y}=(y_1,y_2,\ldots,y_n)$, an indicator variable for imperfect debugging such that $y_i=1$ if the ith failure is caused by an error due to imperfect debugging and $y_i=0$ if the error is not due to imperfec debugging.

The maximum likelihood and Bayesian approaches to the estimation of the above parameters are summarized below. The details of the procedure are given in [20].

4.1 Maximum Likelihood Method

The likelihood function for N, p and λ is

$$L(N,p,\lambda|\underline{t},\underline{y})=\prod_{i=1}^{n}\{N-p(i-1)\}e^{-\{N-p(i-1)\}\lambda t_i}\cdot\left\{\frac{q(i-1)}{N-p(i-1)}\right\}^{y_i}\cdot\left\{\frac{N-(i-1)}{N-p(i-1)}\right\}^{1-y_i}$$

The maximum likelihood estimates ($\hat{N}, \hat{p}$ and $\hat{\lambda}$) are obtained as solutions of the simultaneous non-linear equations

$$\lambda \sum_{i=1}^{n} t_i = \sum_{i=1}^{n} \frac{1-y_i}{N-(i-1)}$$

$$\lambda \sum_{i=1}^{n} (i-1)t_i = \sum_{i=1}^{n} y_i/q$$

$$n/\lambda = \sum_{i=1}^{n} \{N-p(i-1)\}t_i \ .$$

The joint 100(1-α)% confidence regions for N, p and λ are obtained from

$$\ell(\hat{N},\hat{p},\hat{\lambda}|\underline{t},\underline{y})-\ell(N,p,\lambda|\underline{t},\underline{y}) = \frac{1}{2}\chi^2_{3;\alpha} \ ,$$

where

$$\ell(\cdot|\underline{t},\underline{y}) \equiv \log L(\cdot|t;\underline{y}) \ .$$

The estimated variance-covariance matrix for $\hat{N}$, $\hat{p}$ and $\hat{\lambda}$ is

$$\hat{\Sigma}_{cov} = \left.\begin{bmatrix} r_{NN} & r_{Np} & r_{N\lambda} \\ r_{pN} & r_{pp} & r_{p\lambda} \\ r_{\lambda N} & r_{\lambda p} & r_{\lambda\lambda} \end{bmatrix}^{-1}\right|_{\substack{N=\hat{N}\\ p=\hat{p}\\ \lambda=\hat{\lambda}}}$$

where

$$r_{NN} = \sum_{i=1}^{n} 1/\{N-(i-1)\}\{N-p(i-1)\}$$

$$r_{Np} = r_{pN} = 0$$

$$r_{N\lambda} = r_{\lambda N} = \frac{1}{\lambda} \sum_{i=1}^{n} 1/\{N-p(i-1)\}$$

$$r_{pp} = \begin{cases} \frac{1}{q} \sum_{i=1}^{n} (i-1)/\{N-p(i-1)\} & \text{if } q \neq 0 \\ \infty & \text{if } q=0 \,. \end{cases}$$

$$r_{p\lambda} = r_{\lambda p} = -\frac{1}{\lambda} \sum_{i=1}^{n} (i-1)/\{N-p(i-1)\}$$

$$r_{\lambda\lambda} = n/\lambda^2 \,.$$

4.2 Bayesian Inference

Now we describe a Bayesian approach for obtaining posterio point estimates and the highest posterior density (HPD) region fc parameters N, p and λ .

The choice of the prior distribution for a parameter is governed by several factors. In our case we take the conjugate priors, which for N and λ are gamma distributions while for p it is a beta distribution, i.e.,

$$P(N) \propto N^{\alpha-1} e^{-\beta N} \,, \qquad N>0$$

$$P(p) \propto p^{\pi-1} (1-p)^{\rho-1} \,, \qquad 0 \le p \le 1$$

$$P(\lambda) \propto \lambda^{\mu-1} e^{-\gamma\lambda} \,, \qquad \lambda>0 \,.$$

By applying Bayes theorem the joint posterior distributic of N, p and λ for given priors and the data is obtained as

$$p(N,p,\lambda|\underline{t},\underline{y}) \propto p(N,p,\lambda) L(N,p,\lambda|\underline{t},\underline{y}) \,.$$

Let $\hat{N}$, $\hat{p}$ and $\hat{\lambda}$ be the Bayesian point estimates for N, p and λ, respectively. That is, the point $(\hat{N},\hat{p},\hat{\lambda})$ is the mode of the joint posterior distribution $p(N,p,\lambda|\underline{t},\underline{y})$, i.e. it attains its maximum at $(\hat{N},\hat{p},\hat{\lambda})$. Then, $\hat{N}$, $\hat{p}$ and $\hat{\lambda}$ are the values that satisfy

$$-\lambda \sum_{i=1}^{n} t_i + \sum_{i=1}^{n} \frac{1-y_i}{N-(i-1)} + \frac{\alpha-1}{N} - \beta = 0 ,$$

$$\lambda \sum_{i=1}^{n} (i-1)t_i - \Sigma y_i/(1-p) + \frac{\pi-1}{p} - \frac{\rho-1}{1-p} = 0 ,$$

and

$$n/\lambda - \sum_{i=1}^{n} \{N-p(i-1)\}t_i + \frac{\mu-1}{\lambda} - \gamma = 0 .$$

Finally, the $100(1-\alpha)\%$ Bayesian confidence region is given by

$$f(N,p,\lambda) = C ,$$

where

$$\begin{aligned} f(N,p,\lambda) = {} & n\log\lambda - \sum_{i=1}^{n} \{N-p(i-1)\}t_i \\ & + \sum_{i=1}^{n} y_i \log(1-p) + \sum_{i=1}^{n} (1-y_i)\log\{N-(i-1)\} \\ & + (\alpha-1)\log N - \beta N \\ & + (\mu-1)\log\lambda - \gamma\lambda \\ & + (\pi-1)\log p + (\rho-1)\log(1-p) \end{aligned}$$

and

$$C = f(\hat{N},\hat{p},\hat{\lambda}) - \frac{1}{2}\chi^2_{3;\alpha} .$$

5. BAYESIAN SOFTWARE CORRECTION LIMIT POLICIES

The objective of the investigation was to provide an optimum correction limit policy for a large-scale software system subject to random error occurrences and error removals in an operational phase. When an error occurs a corrective action is undertaken to remove it. Such an action can be scheduled at two levels, which we call Phase I and Phase II. By Phase I we mean that the corrective action will be undertaken by the programmer while Phase II action is undertaken by a system analyst or system designer. First, Phase I corrective action is scheduled for a specified time T. If the error is not corrected in this time, it is referred to Phase II. This sequence of corrective actions in an operational phase is shown in Figure 5.1. Our objective is to determine the optimum value T^* of T which minimizes the long run average cost. Two models are developed for this purpose. In the first model we assume that the cost of observations of error occurrence and correction time, prior to the implementation of the optimum policy, is negligible. The second model incorporates the cost of observations.

Details of the model and related results are given in reference [8].

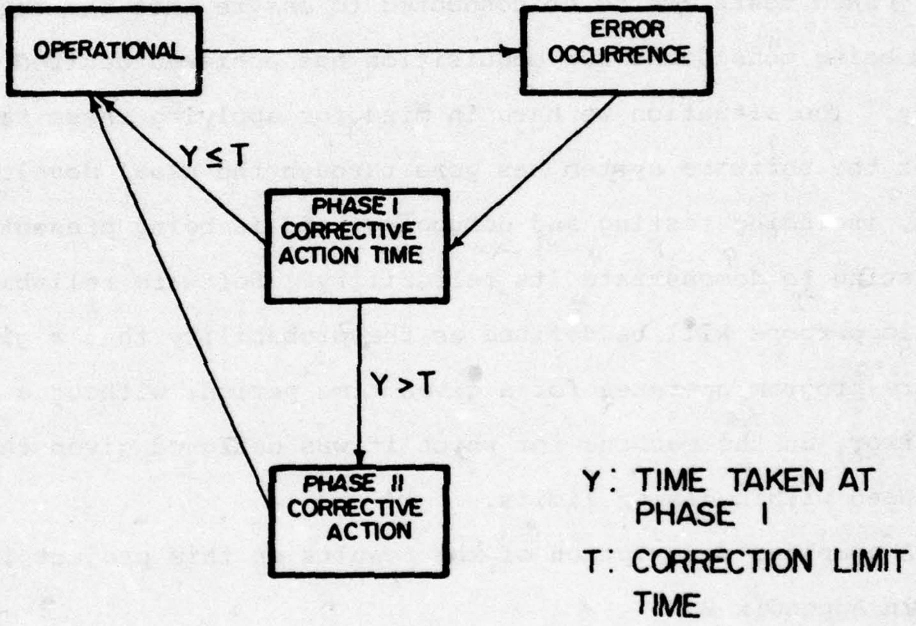


Figure 5.1 Sequence of Corrective Actions in Operational Phase

6. SOFTWARE RELIABILITY DEMONSTRATION TEST PLANS

The purpose of this task was to describe the theory, methodology, and procedures for software reliability demonstration tests. Such tests are to be conducted to ensure that the software system being considered for acquisition has achieved desired reliability. The situation we have in mind for applying these tests is that the software system has gone through the usual development phases, including testing and debugging, and is being presented for testing to demonstrate its reliability. Software reliability for this purpose will be defined as the probability that a given software program operates for a given time period, without a software error, on the machine for which it was designed given that it is used within design limits.

A complete description of the results on this project is given in Appendix B.

7. COMPUTER PROGRAMS

The computer programs required for computations of the quantities described in Sections 2 through 6 are given in Appendix C. The programs are self-explanatory, give the list of input/output parameters and include listings of the needed subroutines.

SELECTED REFERENCES

[1] Akiyama, F. (1971), "An Example of Software Debugging," 1971 IFIP Congress, pp. TA-3-37 to TA-3-42.

[2] Baker, W. F. (1977), "Software Data Collection and Analysis: A Real-Time System Project History," IBM Corporation, Final Technical Report, RADC-TR-77-192, June 1977.(A041644)

[3] Belady, L. A. and Lehman, M. M. (1976), A Model of Large Program Development, IBM Systems Journal, Vol. 15, no. 3 pp. 225-252.

[4] Boehm, B. W., Brown, J. R., and Lipow, M. (1976), "Quantitative Evaluation of Software Quality," Proc. 2nd International Conference on Software Engineering, pp. 54-59.

[5] Endres, A. (1975), "An Analysis of Errors and Their Causes in System Programs," Proceedings: 1975 International Conference on Reliable Software, pp. 327-336.

[6] Fries, M. J. (1977), "Software Error Data Acquisition," Boeing Aerospace Company, Final Technical Report, RADC-TR-77-130, April 1977.(A039916)

[7] Goel, A. L. and Okumoto, K. (1978), "An Imperfect Debugging Model for Reliability and Other Quantitative Measures of Software Systems," Technical Report No. 78-1, Department of IE & OR, Syracuse University.

[8] Goel, A. L. and Okumoto, K. (1978), "Bayesian Software Correction Limit Policies," Technical Report No. 78-4, Department of IE & OR, Syracuse University.

[9] Goel, A. L. and Okumoto, K. (1978), A Time Dependent Error Detection Rate Model for a Large-scale Software System," to appear in the Third USA-Japan Computer Conference Proceedings.

[10] Halstead, M. H. (1977), Elements of Software Science, Elsevier.

[11] Jelinski, J. and Moranda, P. B. (1972), "Software Reliability Research," in Statistical Computer Performance Evaluation, edited by W. Freiberger, Academic Press.

[12] Jelinski, J. and Moranda, P. B. (1973), "Applications of a Probability-Based Model to a Code Reading Experiment," Record: 1973 IEEE Symposium on Computer Software Reliability, pp. 78-81.

[13] Jones, T. C. (1978), "Measuring Programming Quality and Productivity," IBM Systems Journal, Vol. 17, No. 1, pp. 39-63.

[14] Littlewood, B. (1975), "A Reliability Model for Markov Structured Software," Applied Statist., Vol. 24, pp. 172-177.

[15] Littlewood, B. and Verrall, J. L. (1973), "A Bayesian Reliability Growth Model for Computer Software," Applied Statist., Vol. 22, pp. 332-346.

[16] Miyamoto, I. (1975), "Software Reliability in On-Line Real Time Environment," Proceedings: 1975 International Conference on Reliable Software, pp. 194-203.

[17] Moranda, P. (1975), "A Comparison of Software Error-rate Models," 1975 Texas Conference on Computing.

[18] Motley, R. W. et al (1977), "Statistical Prediction of Programming Errors," IBM Corporation, Final Technical Report, RADC-TR-77-175, May 1977.(A041106)

[19] Musa, J. D. (1975), "A Theory of Software Reliability and its Application," IEEE Trans. on Software Engineering, Vol. SE-1, No. 3, pp. 312-327.

[20] Okumoto, K. and Goel, A. L. (1978), "Classical and Bayesian Inference for the Software Imperfect Debugging Model," Technical Report No. 78-2, Department of IE & OR, Syracuse University.

[21] Okumoto, K. and Goel, A. L. (1978), "Availability Analysis of Software Systems under Imperfect Maintenance," Technical Report No. 78-3, Department of IE & OR, Syracuse University.

[22] Ross, S. M. (1970), Applied Probability Models With Optimization Applications, Holden-Day.

[23] Roussas, G. G. (1973), A First Course in Mathematical Statistics, Addison-Wesley.

[24] Rye, P. et al (1977), "Software Systems Development: A CSDL Project History," The Charles Stark Draper Laboratory, Inc., Final Technical Report, RADC-TR-77-213, June 1977. (A042186)

[25] Schick, G. J. and Wolverton, R. W. (1972), "Assessment of Software Reliability," TRW Systems Group, TRW Software Series, TRW-SS-72-04, September 1972.

[26] Schneidewind, N. J. (1972), "An Approach to Software Reliability Prediction and Quality Control," AFIPS Conference Proceedings, Vol. 41 Part II, Fall Joint Computer Conference, pp. 837-838.

[27] Schneidewind, N. J. (1975), "Analysis of Error Processes in Computer Software," Proceedings: 1975 International Conference on Reliable Software, pp. 337-346.

[28] Shooman, M. L. (1972), "Probabilistic Models for Software Reliability Prediction," Statistical Computer Performance Evaluation, pp. 485-502, Academic Press, New York.

[29] Shooman, M. L. and Bolsky, M. I. (1975), "Types, Distribution, and Test Correction Times for Programming Errors," Proceedings: 1975 International Conference on Reliable Software, pp. 347-357.

[30] Sukert, A. N. (1977), "An Investigation of Software Reliability Models," Proc. 1977 R & M.

[31] Sukert, A. N. (1977), "A Multi-Project Comparison of Software Reliability Models," Computers in Aerospace Conference, pp. 413-421.

[32] Thayer, T. A. et al (1976), "Software Reliability Study," TRW Defense & Space Systems Group, Final Technical Report, RADC-TR-76-238, Aug. 1976.(A030798)

[33] Trivedi, A. K. and Shooman, M. L. (1975), "A Many-State Markov Model for the Estimation and Prediction of Computer Software Performance Parameters," Proceedings: 1975 International Conference on Reliable Software, pp. 208-220.

[34] Wagoner, W. L. (1973), "The Final Report on Software Reliability Measurement Study," Aerospace Report No. TOR-0074(4112)-1.

[35] Willman, H. E. Jr. et al (1977), "Software Systems Reliability: A Raytheon Project History," Raytheon Company, Final Technical Report, RADC-TR-77-188, June 1977.(A040992)

APPENDIX A

AN ANALYSIS OF RECURRENT SOFTWARE ERRORS IN A REAL-TIME CONTROL SYSTEM

In this Appendix we present an analysis of software error data from a large-scale software project using the imperfect debugging model discussed in Section 2. The model parameters are estimated from the data and the values predicted from the model are compared with the observed values. Joint confidence regions for the parameters are also constructed which permit a study of the sensitivity of predictions.

A.1 Analysis of Error Data from a Real Time Control System

A real-time control system for a land-based radar system was developed by Raytheon Co. in a modular fashion. Nearly all of the modules were written in JOVIAL/J3. The error data base was extracted from 2165 Software Problem Reports (SPRs) written against 109 operational software modules over the development phases and is described in [35]. Table A.1 shows the distribution of the SPRs by month opened during a 22 month period of integration, acceptance and operational testing phases.

The available data give the number of total errors (u_i) and the number of imperfect debugging or recurrent errors (v_i) detected by time t_i. The parameters under consideration are the initial number of software errors (N), the error occurrence rate for each error (λ), and the probability (p) of perfect debugging. We first estimate the parameters N, p and λ from these data ($\underset{\sim}{t}, \underset{\sim}{u}, \underset{\sim}{v}$) and then compare the results obtained from IDM with the observed values.

TABLE A.1

ERROR DATA BY MONTH

Month	Total Number of Errors		Imperfect Debugging Errors	
(t_i)	$u_i - u_{i-1}$	u_i	$v_i - v_{i-1}$	v_i
1	122	122	6	6
2	98	220	1	7
3	82	302	1	8
4	75	377	3	11
5	113	490	2	13
6	85	575	3	16
7	105	680	2	18
8	47	727	1	19
9	61	788	1	20
10	25	813	1	21
11	28	841	0	21
12	42	883	1	22
13	18	901	0	22
14	17	918	0	22
15	28	946	0	22
16	14	960	0	22
17	5	965	0	22
18	3	968	0	22
19	3	971	0	22
20	13	984	0	22
21	5	989	0	22
22	10	999	0	22

Using the data in Table A.1 and the results from Section 2, the results are obtained as shown in Table A.2.

The joint confidence regions for N and λ for p=0.974 are plotted in Figure A.1. The plots of the actual and fitted SPR's by month are shown in Figure A.2, and the plots of actual and predicted number of remaining errors are given in Figure A.3.

TABLE A.2

A SUMMARY OF ERROR DATA ANALYSES

Quantities of Interest		Calculated Values
	$\hat{N}$	1079 (Errors)
	$\hat{p}$	0.974
	$\hat{\lambda}$	0.1235 (per month)
	$\hat{a}(\equiv \frac{\hat{N}}{\hat{p}})$	1108 (Errors)
	$\hat{b}(\equiv \hat{p}\hat{\lambda})$	0.1203 (per month)
90% bounds	$\{\underline{N}, \overline{N}\}$	{914, 1244}
90% bounds	$\{\underline{p}, \overline{p}\}$	{0.964, 0.984}
90% bounds	$\{\underline{\lambda}, \overline{\lambda}\}$	{0.096, 0.151}
90% bounds	$\{\underline{a}, \overline{a}\}$	{933, 1283}
90% bounds	$\{\underline{b}, \overline{b}\}$	{0.0926, 0.1480}
	$\hat{\rho}_{\hat{N},\hat{\lambda}}$	-0.723
	$\hat{\rho}_{\hat{a},\hat{b}}$	-0.743

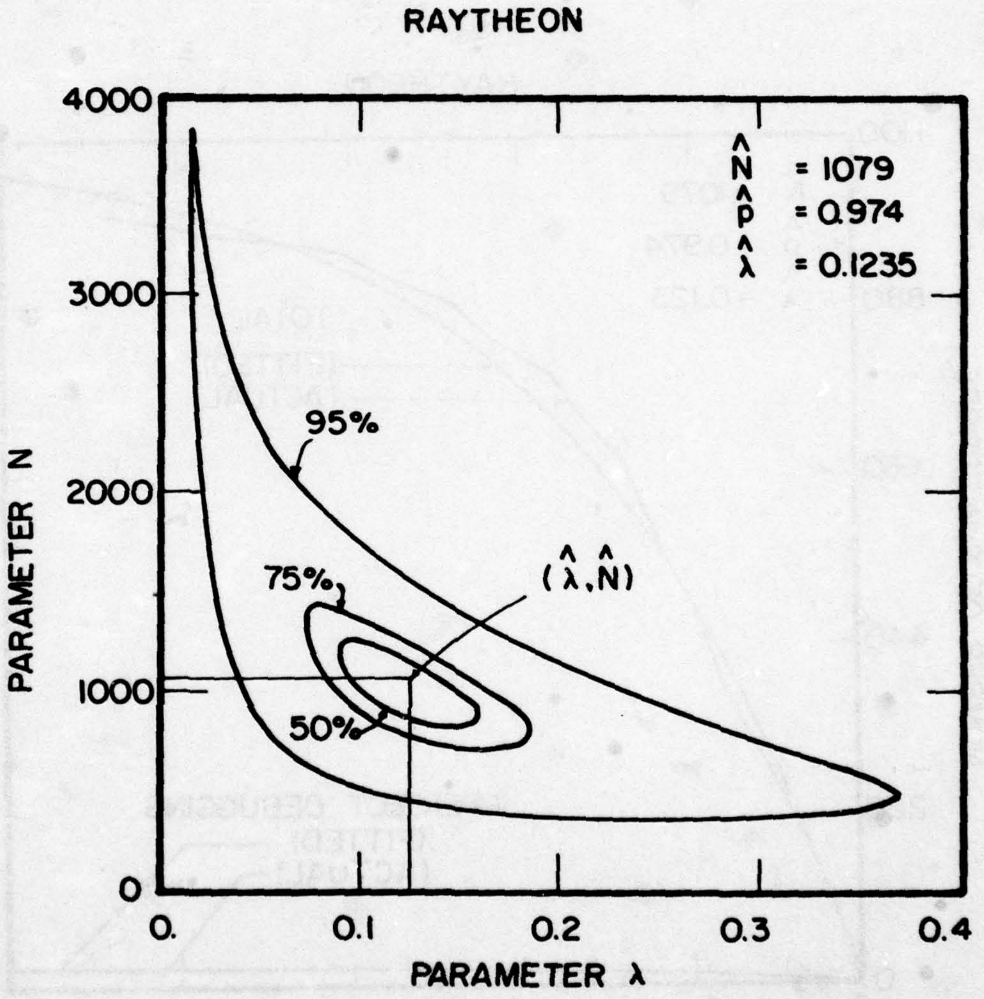


Figure A.1 Joint confidence regions for N and λ for $p=\hat{p}$

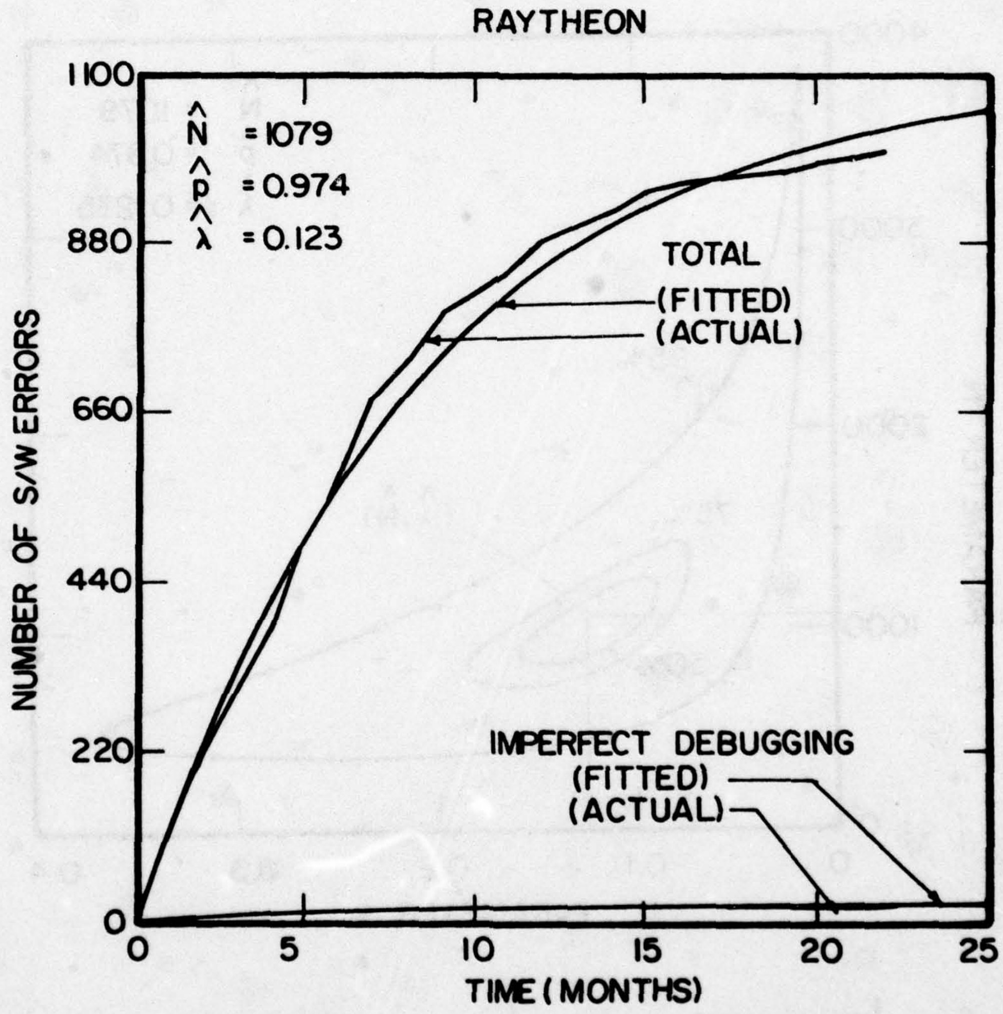


Figure A.2 Actual and fitted SPRs by month

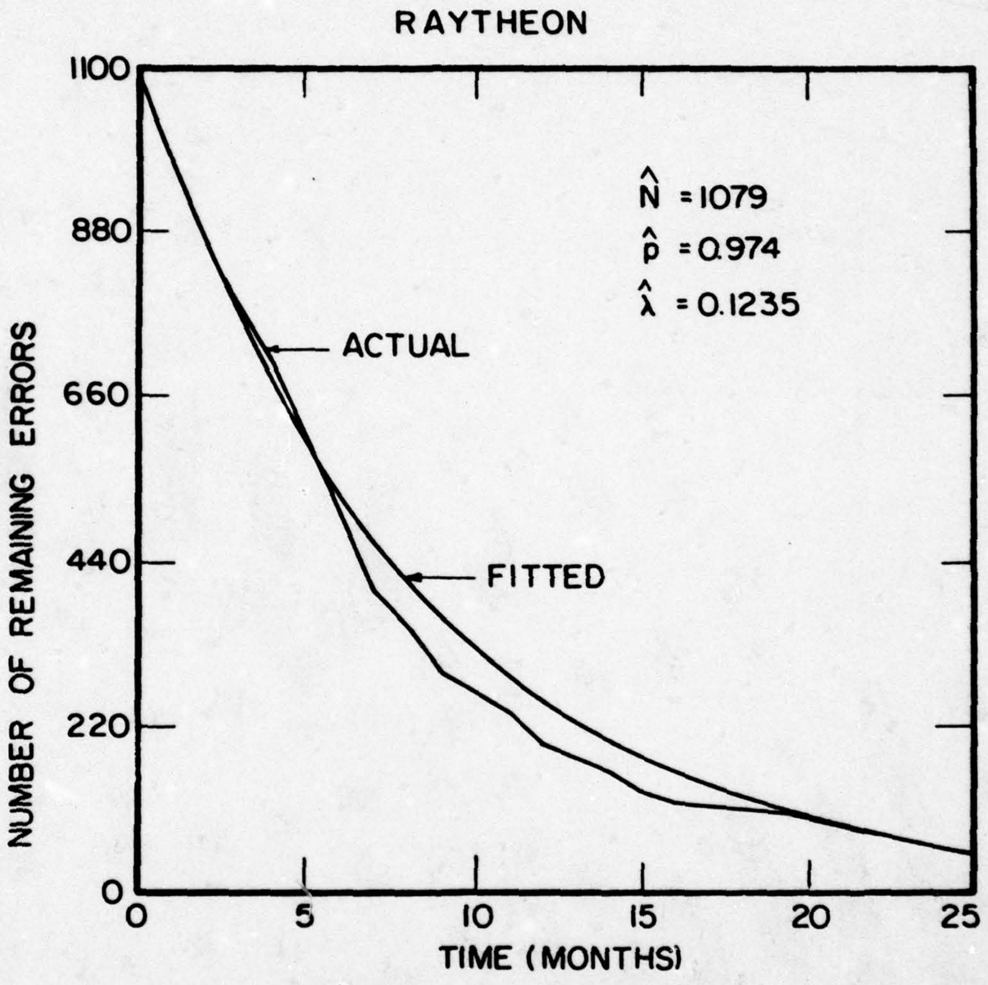


Figure A.3 Plots of the actual and predicted number of remaining errors.

APPENDIX B

SOFTWARE RELIABILITY DEMONSTRATION TEST PLANS

B.1. INTRODUCTION AND PROBLEM DEFINITION

The purpose of this report is to describe the theory, methodology, and procedures for software reliability demonstration tests. Such tests are to be conducted to ensure that the software system being considered for acquisition has achieved desired reliability. The situation we have in mind for applying these tests is that the software system has gone through the usual development phases, including testing and debugging, and is being presented for testing to demonstrate its reliability. Software reliability for this purpose will be defined as the probability that a given software program operates for a given time period, without a software error, on the machine for which it was designed, given that it is used within design limits.

In order to develop an appropriate test plan, we must first choose a model which adequately describes the error occurrence phenomenon. Most software error models are based on the assumption that successive errors follow a decreasing failure rate because of the reduction in the number of remaining errors. However, in this appendix we assume that the times between errors during the demonstration phase follow an exponential distribution, i.e. have a constant failure rate. As a justification for this assumption, we contend that the demonstration time and the number of errors encountered during demonstration will be relatively small and hence a constant failure rate model will be an adequate representation of the error phenomenon. At worst, this model will yield a somewhat conservative test from the viewpoint of DOD.

Let the distribution of error occurrence times be given by

$$f(t|\lambda) = \lambda \exp(-t\lambda), \qquad t \geq 0, \lambda > 0 \qquad (B.1-1)$$

Then the following measures of software reliability can be used interchangeably:

. Reliability, $R(\tau) = P(t > \tau) = e^{-\tau\lambda}$ (B.1-2)

. Meantime Between Software Failure (MTBSF) = $1/\lambda$

. Software Failure Rate (SFR) = λ

In the following we shall take λ as the measure of software reliability.

In the classical set up, the demonstration tests are generally designed to distinguish between two values of λ, namely the maximum acceptable SFR, λ_1 and the specified SFR, λ_0. The decision to accept or reject the software system is based upon test results which are subject to random fluctuation. A loss is incurred when either an accept or a reject decision is wrongly taken.

The loss corresponding to wrong decisions is quantified by two risks called the producer's risk, $\alpha = P(R|\lambda_0)$, and the consumer's risk $\beta = P(A|\lambda_1)$, where A and R denote acceptance and rejection of the software system, respectively.

In this context various types of demonstration plans can be designed to limit α and β to desired values. For example, a truncated single sample plan for the system is employed as follows. The plan consists of using the software in an environment which is representative of the operational environment for a time period T. The number of errors r encountered during this time period is recorded. (In this study we assume that all errors are of the same severity. When errors are classified according to the degree of severity, the problem becomes quite complicated and is beyond the scope of this investigation). If r is less than or equal to a pre-specified number, r*, the software is accepted. Otherwise, the system is rejected. The design of such a plan consists of obtaining the quantities T and r* such that the desired risks α and β are satisfied.

One disadvantage in the above approach is that λ is assumed to be an unknown, fixed constant. In practice there may be sufficient reason to believe that knowledge about λ is available in some quantifiable form and one would like to incorporate such information in the development of the demonstration test plan. Another important situation arises when the risks associated with the demonstration test are not adequately represented by the classical risks (α, β) and interest lies in associating risks with the posterior distribution of λ. In such cases one resorts to a Bayesian approach for test design.

The problem under consderation then, is the design of single sample software reliability demonstration plans when error occurrence times are assumed to follow the exponential distribution.

Before describing the development of the test plans, we first discuss the various risks that arise in the above situations in Section B.2. Fixed time, classical tests are then considered in Section B.3. In Sections B.4 and B.5 Bayesian tests are developed for two situations: (i) λ has a noninformative prior distribution, and (ii) information about λ can be quantified from the testing and debugging data.

Section B.6 presents the step-by-step procedure to be employed to demonstrate the properties and performance of the demonstration test plans.

B.2. DEFINITIONS AND INTERPRETATION OF RISKS

The following quantities are of interest

1. The probability $P(R|\lambda = \lambda_0) = \alpha$ that a software system with specified SFR is rejected.
2. The probability $P(A|\lambda = \lambda_1) = \beta$ that a software system with maximum acceptable SFR is accepted.
3. The probability $P(A|\lambda \geq \lambda_1) = \bar{\beta}$ that a software system which is of unacceptable reliability is accepted.
4. The probability $P(R|\lambda \leq \lambda_0) = \bar{\alpha}$ that a software system of acceptable reliability is rejected.
5. The probability $P(\lambda \geq \lambda_1|A) = \beta^*$ that the SFR of a system which has been accepted is more than the maximum acceptable SFR.
6. The probability $P(\lambda \geq \lambda_0|A) = \beta^{**}$ that the SFR of a system which has been accepted is more than the specified SFR.
7. The probability $P(\lambda \leq \lambda_0|R) = \alpha^*$ that the SFR of a system which has been rejected is less than the specified SFR.
8. The probability $P(R)$ that the software system is rejected.
9. The probability $P(A)$ that the software system is accepted.
10. The probability $P(\lambda \geq \lambda')$ that a-priori, the software system has SFR which is more than λ'.

These quantities will now be discussed in some detail.

B. 2.1, <u>Classical Risks (α, β)</u>

The classical producer's risk α and consumer's risk β are defined as follows:

$\alpha = P(R|\lambda = \lambda_0)$, the probability of rejecting a software system whose SFR is equal to the specified value, λ_0. (B.2-1)

$\beta = P(A|\lambda = \lambda_1)$, the probability of accepting a software system whose SFR is equal to the maximum acceptable value. (B.2-2)

The (α, β) risks represent two points on the classical operating characteristic (OC) curve which is a plot of $P(A|\lambda)$ versus λ. These risks do not provide an explicit control of the probability of acceptance for values of λ other than λ_1 and λ_0. However, $P(A|\lambda)$ decreases monotonically with λ. Hence, if $\lambda < \lambda_0$, the probability of rejection is less than α. If $\lambda > \lambda_1$, the probability of acceptance is less than β. The shape of the OC curve governs the degree of protection provided in the indifference zone between λ_1 and λ_0.

B. 2.2. <u>Average Risks ($\bar{\alpha}, \bar{\beta}$)</u>

The average risks are defined as follows

$\bar{\alpha} = P(R|\lambda \leq \lambda_0)$, the probability of rejecting a software system with a SFR less than or equal to the specified value, λ_0. (B.2-3)

$\bar{\beta} = P(A|\lambda \geq \lambda_1)$, the probability of accepting a software system with a SFR greater than or equal to λ_1. (B.2-4)

Mathematically, the risks may be expressed as:

$$P(R|\lambda \leq \lambda_0) = \frac{P(R,\lambda \leq \lambda_0)}{P(\lambda \leq \lambda_0)} = \frac{\int_0^{\lambda_0} P(R|\lambda)p(\lambda)d\lambda}{\int_0^{\lambda_0} p(\lambda)d\lambda} \qquad \text{(B.2-5)}$$

or

$$\bar{\alpha} = \int_0^{\lambda_0} P(R|\lambda)\cdot p(\lambda|\lambda \leq \lambda_0)d\lambda \qquad \text{(B.2-6)}$$

and, similarly

$$\bar{\beta} = \int_{\lambda_1}^{\infty} P(A|\lambda)\cdot p(\lambda|\lambda \geq \lambda_1)d\lambda \qquad \text{(B.2-7)}$$

The average risks provide the following protection. If the producer produces a large number of software systems, then, in the long run, less than $\bar{\alpha}$ percent of the desired ones will be rejected. If the consumer buys a large number of software systems, then, in the long run, less than 100 $\bar{\beta}$ percent of the bad systems will be accepted. No explicit control on the probability of acceptance is provided at any specific value of λ when we use the average risk criteria.

2.3 Posterior Risks (α^*, β^*)

The (α^*, β^*) risks are defined as follows

$$\alpha^* = P(\lambda \leq \lambda_0 | R)$$

This risk is the long run probability of a rejected software system being good.

$$\beta^* = P(\lambda \geq \lambda_1 | A)$$

This risk is the long run probability of an accepted system being bad. Mathematically,

$$\alpha^* = P(\lambda \leq \lambda_0 | R) = \int_0^{\lambda_0} P(\lambda | R) d\lambda = \frac{\int_0^{\lambda_0} P(R|\lambda)\ p(\lambda) d\lambda}{\int_0^{\infty} P(R|\lambda) p(\lambda) d\lambda}$$

$$\beta^* = P(\lambda \geq \lambda_1 | A) = \int_{\lambda_1}^{\infty} P(\lambda | A) d\lambda = \frac{\int_{\lambda_1}^{\infty} P(A|\lambda)\ p(\lambda) d\lambda}{\int_0^{\infty} P(A|\lambda)\ p(\lambda) d\lambda}$$

These risks can also be interpreted in a "degree of belief" sense. Thus, α^* would represent a persons's degree of belief that if a software system has been rejected, it has a SFR which is better than the specified value. Similarly, β^* would be the degree of belief that the SFR of an accepted system is worse than the maximum acceptable value.

B. 2.4 Probability of Rejection P(R)

This is a single number given by

$$P(R) = \int_0^\infty P(R|\lambda)p(\lambda)d\lambda \qquad \text{(B.2-12)}$$

or

$$P(R) = 1 - P(A) = 1 - \int_0^\infty P(A|\lambda)p(\lambda)d\lambda \qquad \text{(B.2-13)}$$

Note that the integration is over the entire range of λ and specification of λ_0, which is usually specified in conjunction with a producer's risk, is unnecessary.

In the frequency sense we have

$$P(R) = \frac{\text{Total number of systems rejected}}{\text{Total number of systems tested}}$$

For the producer this criterion implies that, in the long run, less than (100)·P(R) percent of the systems will be rejected.

B.2.5. Alternate Posterior Consumer's Risk β^{**}

We define a new risk associated with the posterior distribution of λ as follows

$$\beta^{**} = P(\lambda \geq \lambda_0 | A) = \int_{\lambda_0}^{\infty} f(\lambda|A)d\lambda, \tag{B.2-14}$$

Where $f(\lambda|A)$ is the pdf of λ conditional on acceptance. This can be written as

$$\beta^{**} = \frac{\int_{\lambda_0}^{\infty} P(A|\lambda)\ p(\lambda)d\lambda}{\int_{0}^{\infty} P(A|\lambda)\ p(\lambda)d\lambda} . \tag{B.2-15}$$

This risk gives the long run probability of the accepted system having a λ above the specified SFR λ_0.

B. 3. DESIGN OF TEST PLAN FOR CLASSICAL RISKS

Now we consider the design of a single sample plan where the parameters are T and r*. Since the time to software failure is exponential, the observed number of software errors r in fixed time T has a Poisson distribution, i.e.

$$f(r|\lambda) = \frac{e^{-T\lambda} (T\lambda)^r}{r!} \tag{B.3-1}$$

The two risks can be written as:

$$\sum_{r=0}^{r^*} \frac{e^{-T\lambda_0} (T\lambda_0)^r}{r!} = 1 - \alpha \tag{B.3-2}$$

and

$$\sum_{r=0}^{r^*} \frac{e^{-T\lambda_1} (T\lambda_1)^r}{r!} = \beta \tag{B.3-3}$$

Given λ_1, λ_0, α and β, the above equations can be simultaneously solved to obtain software test time T and allowable number of errors r*.

Individual specification of λ_1 and λ_0 is not necessary. Let $K = \lambda_1/\lambda_o$, and let $T^* = T\lambda_0$. Then, to satisfy the stated risks we can write

$$\sum_{r=0}^{r^*} \frac{e^{-T^*} (T^*)^r}{r!} \geq 1 - \alpha \qquad \text{(B,3-4)}$$

and

$$\sum_{r=0}^{r^*} \frac{e^{-KT^*} (KT^*)^r}{r!} \leq \beta \qquad \text{(B.3-5)}$$

Given (K, α, β) the above two equations can be solved to obtain (T^*, r^*). These equations can be solved numerically or by using tables of cumulative Poisson probabilities. Clearly, test plans with identical λ_0/λ_1 have the same T^* and r^* values. Given the specified value λ_0, the actual test time is obtained as $T = T^*/\lambda_0$.

4. BAYESIAN TEST PLANS FOR NONINFORMATIVE PRIOR DISTRIBUTION

In this section we consider the case when it is possible to express the unknown parameter λ in terms of a metric $\phi(\lambda)$ so that the corresponding likelihood is data translated. This means that the likelihood function for $\phi(\lambda)$ is completely determined a-priori except for its location which depends on the software failure data yet to be observed. This state of indifference can be expressed by taking $\phi(\lambda)$ to be locally uniform, and the resulting prior distribution is called noninformative for $\phi(\lambda)$ with resepect to data. A more detailed discussion on noninformative prior distribution can be found in Box and Tiao [1]* . In our case, a noninformative prior for λ is

$$p(\lambda) \propto \lambda^{-1/2} \qquad \text{(B.4-1)}$$

Now, if T is the test time for a software system, the number of errors in T will be given by a Poisson distribution with parameter $T\lambda$ as mentioned earlier. Letting $\Lambda = T\lambda$, the prior for Λ can be written as

$$P(\Lambda) \propto \Lambda^{-1/2} \qquad \text{(B.4-2)}$$

*References at the end of this Appendix

The joint probability distribution of Λ and r is:

$$p(r,\Lambda) = p(r|\Lambda)\cdot p(\Lambda)$$

$$\propto \frac{(\Lambda)^r \cdot e^{-\Lambda}}{r!} \cdot \Lambda^{-1/2}$$

or

$$p(r,\Lambda) = b' \frac{\Lambda^{r-1/2} \cdot e^{-\Lambda}}{r!} \qquad \text{(B.4-3)}$$

where b' is the normalizing constant.

To get the marginal distribution of r, we let Λ be defined over the range (0,d) where d is sufficiently large and $d < \infty$. Then we have

$$p(r) = \int_0^\infty P(r,\Lambda)\cdot d\Lambda = \int_0^\infty b' \cdot \frac{\Lambda^{r-1/2} \cdot e^{-\Lambda}}{r!} d\Lambda \qquad \text{(B.4-4)}$$

Now we choose d such that

$$\int_0^\infty \frac{b' \Lambda^{r-1/2} \cdot e^{-\Lambda}}{r!} d\Lambda < \varepsilon \qquad \text{(B.4-5)}$$

for some given sufficiently small $\varepsilon > 0$. Then

$$p(r) = \frac{b!}{r!} \Gamma(r + \frac{1}{2}) \qquad \text{(B.4-6)}$$

and

$$P(A) = \sum_{r=0}^{r^*} p(r) = \sum_{r=0}^{r^*} \frac{b'\Gamma(r + \frac{1}{2})}{r!} \qquad \text{(B.4-7)}$$

In order to get expressions for various risks, we proceed as follows.

Let $\Lambda_0 = T\lambda_0$. Then from (B.4-3) we get,

$$P(\lambda \leq \lambda_0, \text{ accept}) = P(\Lambda \leq \Lambda_0, \text{ accept})$$

$$= \sum_{r=0}^{r^*} \int_0^{\Lambda_0} b' \cdot \frac{\Lambda^{r-1/2} \cdot e^{-\Lambda}}{r!} \, d\Lambda \qquad \text{(B.4-8)}$$

Substituting the above expressions into the appropriate formulae in Section 2, we get the equations for desired risks. For example, if we are interested in the risk combination $(\bar{\alpha}, \beta^*)$, we get from (B.2-3) and (B.2-9), respectively:

$$\bar{\alpha} = 1 - P(A | \lambda \leq \lambda_0)$$

$$= 1 - \frac{P(A, \Lambda \leq \lambda_0)}{P(\lambda \leq \lambda_0)}$$

$$= 1 - \frac{\sum_{r=0}^{r^*} \int_0^{\Lambda_0} b' \cdot \frac{\Lambda^{r-1/2} \cdot e^{-\Lambda}}{r!} \, d\Lambda}{\int_0^{\Lambda_0} b' \, \Lambda^{-1/2} d\Lambda}$$

or

$$\bar{\alpha} = 1 - \frac{\sum_{r=0}^{r^*} \int_0^{\Lambda_0} \frac{\Lambda^{r-1/2} \cdot e^{-\Lambda}}{r!} \, d\Lambda}{\int_0^{\Lambda_0} \Lambda^{-1/2} d\Lambda} \qquad \text{(B.4-9)}$$

and

$$\beta^* = P(\lambda \geq \lambda_1 | A)$$

$$= 1 - P(\lambda \leq \lambda_1 | A)$$

$$= 1 - \frac{\sum_{r=0}^{r^*} \int_0^{\Lambda_1} b' \, \Lambda^{r-1/2} \cdot e^{-\Lambda} d\Lambda}{\sum_{r=0}^{r^*} \frac{b' \, \Gamma(r + \frac{1}{2})}{r!}}$$

or

$$\beta^* = 1 - \frac{\sum_{r=0}^{r^*} \int_0^{\Lambda_1} \Lambda^{r-1/2} \cdot e^{-\Lambda} d\Lambda}{\sum_{r=0}^{r^*} \frac{\Gamma(r + \frac{1}{2})}{r!}} \tag{B.4-10}$$

If the consumer's risk of interest is β^{**}, then from (B.2-14) we have

$$\beta^{**} = P(\lambda \geq \lambda_0 | A)$$

$$= 1 - P(\lambda \leq \lambda_0 | A)$$

$$= 1 - \frac{\sum_{r=0}^{r^*} \int_0^{\Lambda_0} \Lambda^{r-1/2} \cdot e^{-\Lambda} \, d\Lambda}{\sum_{r=0}^{r^*} \frac{\Gamma(r + \frac{1}{2})}{r!}} \tag{B. 4-11}$$

The design plan T and r* is obtained by simultaneously solving two equations, one each for the producer's and the consumer's risk. The proper choice of risk combinations will be governed by the protection desired and agreed upon by the vendor and the buyer of the software system or systems.

Two nomograms for the design of plans for $(\bar{\alpha},\beta^*)$ and $(\bar{\alpha},\beta^{**})$ are given on the following pages,

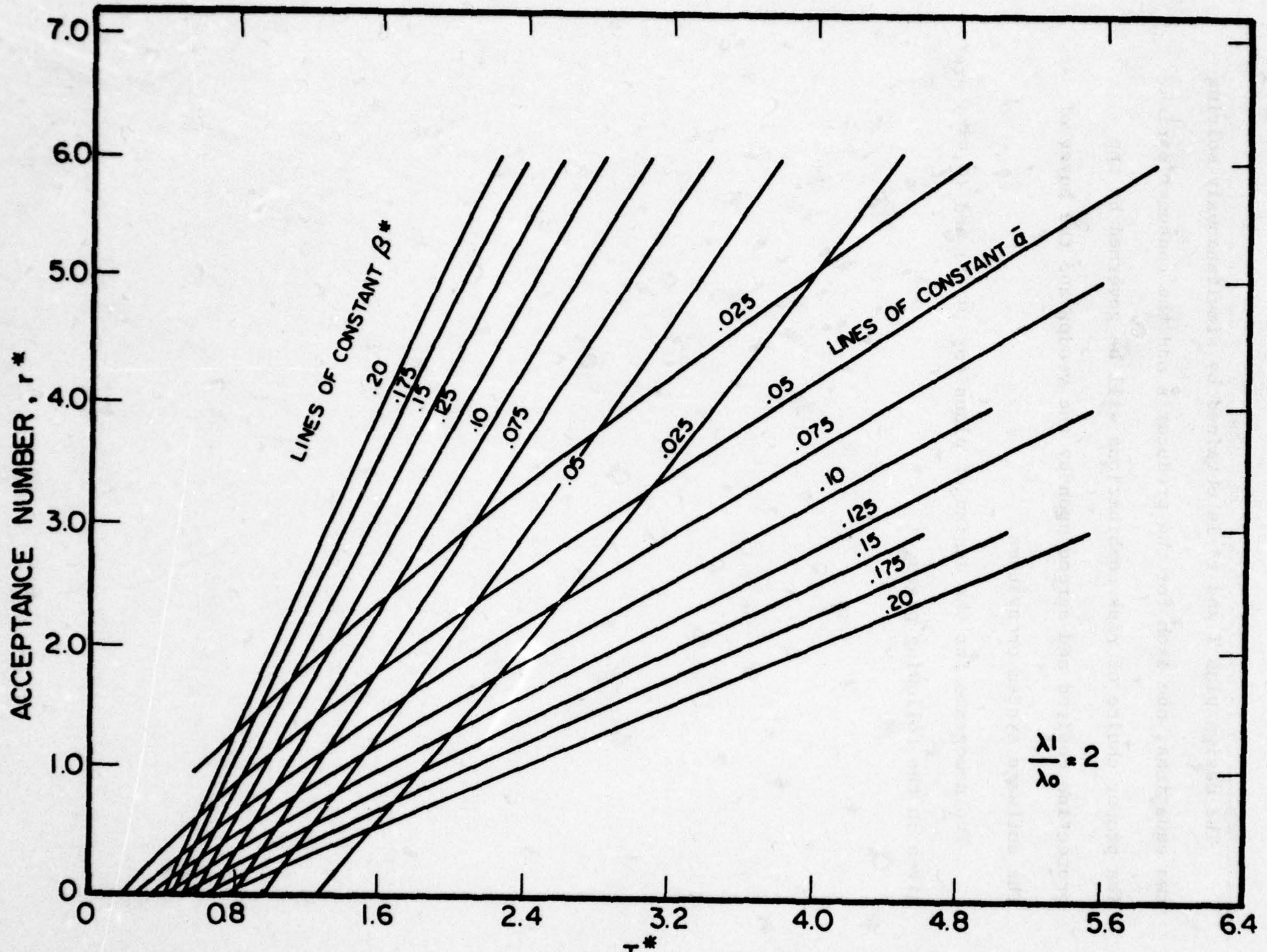


FIG. B-1 COUNTOURS OF $\bar{a}, \beta^*$ FOR DESIGN OF TEST PLANS

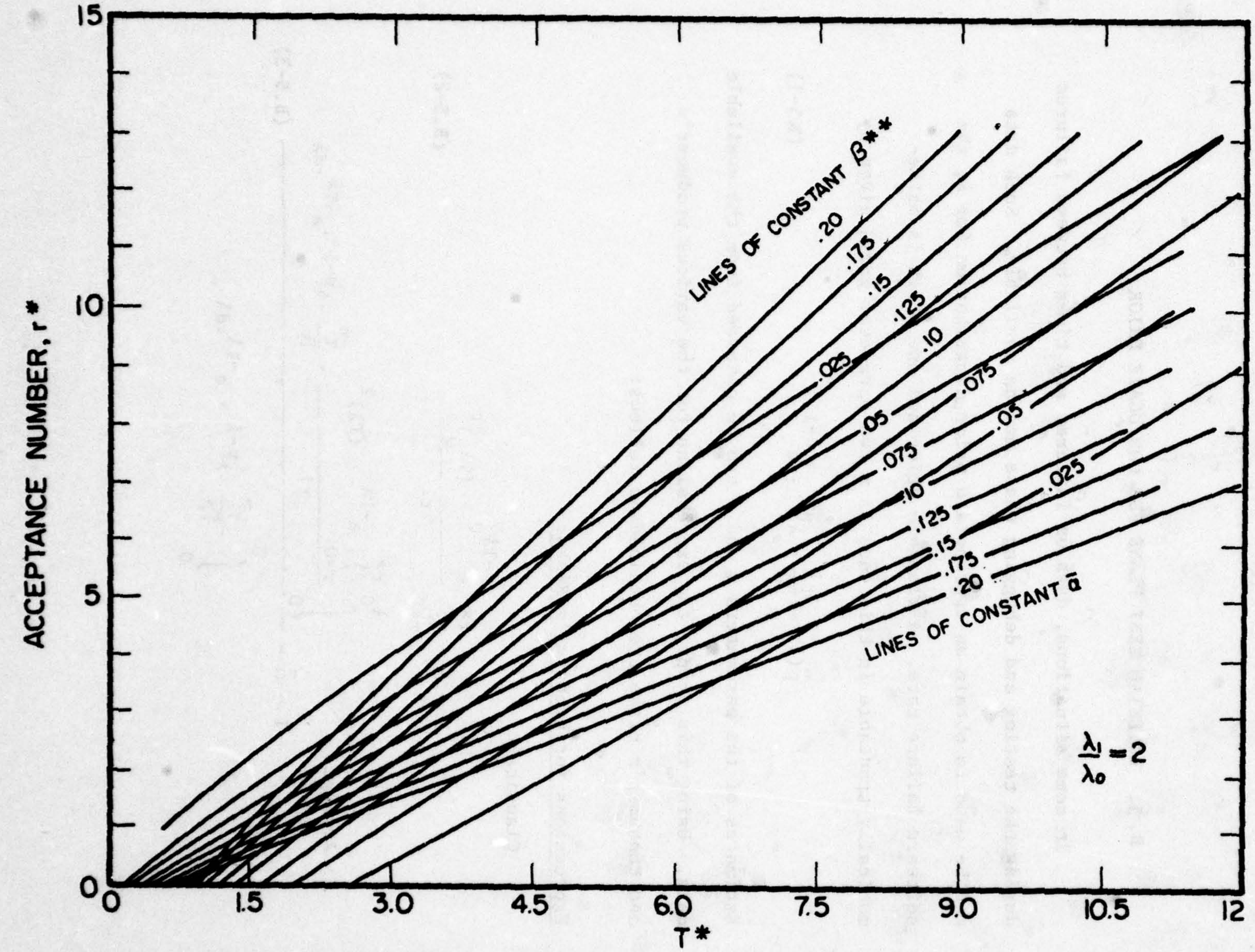


FIG. B-2 CONTOURS OF $\bar{\alpha}, \beta^{**}$ FOR DESIGN OF TEST PLANS

B.5. BAYESIAN TEST PLANS FOR CONJUGATE PRIOR

In some situations, data on failures and times between failures during the testing and debugging phase may be available. Such data can be used to obtain an appropriate prior distribution for λ, the software failure rate. A flexible prior and one which is mathematically tractable in this case is a two parameter gamma given by

$$p(\lambda) = \frac{\tau^{p}}{\Gamma p} \cdot \lambda^{pT} \cdot e^{-\tau\lambda} \qquad \text{(B.5-1)}$$

Estimates of the parameters p and τ can be obtained from the available data. Using this prior, the expressions for the various producer's and consumer's risks are obtained as follows:

<u>Expressions for Producer's Risks:</u>

Classical:

$$1 - \alpha = \sum_{r=0}^{r^*} \frac{e^{-T\lambda_0} \cdot (T\lambda_0)^{r}}{r!} \qquad \text{(B.5-2)}$$

Average:

$$1 - \bar{\alpha} = \frac{\int_0^{\lambda_0} \frac{\sum_{r=0}^{r^*} e^{-T\lambda} \cdot (T\lambda)^2}{r!} \cdot \frac{\tau^p}{\Gamma p} \cdot \lambda^{p-1} \cdot e^{-\tau\lambda} \cdot d\lambda}{\int_0^{\lambda_0} \frac{\tau^p}{\Gamma p} \cdot \lambda^{p-1} \cdot e^{-\tau\lambda} \cdot d\lambda} \qquad \text{(B.5-3)}$$

Posterior:

$$1 - \alpha^* = \frac{\int_0^{\lambda_0} \{1 - \sum_{r=0}^{r^*} \frac{e^{-T\lambda} \cdot (T\lambda)^r}{r!}\} \cdot \frac{\tau^p}{\Gamma p} \cdot \lambda^{p-1} \cdot e^{-\tau\lambda} \cdot d\lambda}{1 - \int_0^{\infty} \{\sum_{r=0}^{r^*} \frac{e^{-T\lambda} (T\lambda)^r}{r!}\} \cdot \frac{\tau^p}{\Gamma p} \cdot \lambda^{p-1} \cdot e^{-\tau\lambda} \cdot d\lambda} \qquad \text{(B. 5-4)}$$

Probability of Rejection:

$$P(R) = 1 - \int_0^{\infty} \{\sum_{r=0}^{r^*} \frac{e^{-T\lambda} (T\lambda)^r}{r!}\} \cdot \frac{\tau^p}{\Gamma p} \cdot \lambda^{p-1} \cdot e^{-\tau\lambda} \cdot d\lambda \qquad \text{(B. 5-5)}$$

Expressions for Consumer's Risks:

Classical:

$$\beta = \sum_{r=0}^{r^*} \frac{e^{-TK\lambda_0} \cdot (KT\lambda_0)^r}{r!} \qquad \text{(B. 5-6)}$$

Average:

$$\bar{\beta} = \frac{\int_{\lambda_1}^{\infty} \{\sum_{r=0}^{r^*} \frac{e^{-T\lambda} \cdot (T\lambda)^r}{r!}\} \cdot \frac{\tau^p}{\Gamma p} \cdot \lambda^{p-1} \cdot e^{-\tau\lambda} \cdot d\lambda}{\int_{\lambda_1}^{\infty} \frac{\tau^p}{\Gamma p} \cdot \lambda^{p-1} \cdot e^{-\tau\lambda} \cdot d\lambda} \qquad \text{(B. 5-7)}$$

Posterior:

$$\beta^* = \frac{\int_{\lambda_1}^{\infty} \{\sum_{r=0}^{r^*} \frac{e^{-T\lambda} \cdot (T\lambda)^r}{r!}\} \frac{\tau^p}{\Gamma p} \cdot \lambda^{p-1} \cdot e^{-\tau\lambda} \cdot d\lambda}{\int_{0}^{\infty} \{\sum_{r=0}^{r^*} \frac{e^{-T\lambda} \cdot (T\lambda)^r}{r!}\} \frac{\tau^p}{\Gamma p} \cdot \lambda^{p-1} \cdot e^{-\tau\lambda} \cdot d\lambda} \qquad \text{(B.5-8)}$$

Alternate Posterior:

$$\beta^{**} = \frac{\int_{\lambda_0}^{\infty} \{\sum_{r=0}^{r^*} \frac{e^{-T\lambda} (T\lambda)^r}{r!}\} \frac{\tau^p}{\Gamma p} \cdot \lambda^{p-1} \cdot e^{-\tau\lambda} \cdot d\lambda}{\int_{0}^{\infty} \{\sum_{r=0}^{r^*} \frac{e^{-T\lambda} \cdot (T\lambda)^r}{r!}\} \frac{\tau^p}{\Gamma p} \cdot \lambda^{p-1} \cdot e^{-\tau\lambda} \cdot d\lambda} \qquad \text{(B.5-9)}$$

B. 6. PROCEDURE FOR DEMONSTRATING THE PERFORMANCE OF THE TEST PLANS

This section provides a step-by-step procedure that will be used to demonstrate the properties and performance of the demonstration test plans for a software system as developed in the previous sections. These procedures will be followed in conducting the demonstration on site.

Prior to conducting the demonstration test in practice, the DOD must make sure that the software system has been developed according to specifications and all the stated development phases have been carried out to meet the objectives.

The steps to be followed for a demonstration test are as follows.

1. Decide the risk criteria (producer's and consumer's risks) for demonstration.
2. Choose the values of the two risks.
3. Design a test plan (T,r*) which meets the risks in Step 2 as closely as possible. (Designs will be obtained using computer programs developed for this purpose.)
4. Simulate software errors using the software error simulator.
5. Apply the demonstration test of Step (3) to simulated errors and make accept/reject decisions.
6. Compare the recorded results with theoretical results.

B.7. REFERENCES

[1] Box, G.E.P. and Tiao, G.C. (1973)

Bayesian Inference in Statistical Analysis. Addison-Wesley

[2] Goel, A.L. and Joglekar, A.M. (1976)

"Reliability Acceptance Sampling Plans Based Upon Prior Distribution: Risk Criteria and Their Interpretation," RADC-TR-76-294, Volume II. (A033516)

APPENDIX C
COMPUTER PROGRAMS

C.1 Programs for the Imperfect Debugging Model (Section 2)

Computer programs to compute the following quantities, for given N, p and λ are given in Tables C.1 to C.8.

- Mean and variance of the first passage time from N to n_0 (SUBROUTINE FRSTRT)
- Pdf and cdf of the first passage time from N to n_0 (SUBROUTINE FPT2)
- Probability of the software system having n_0 errors remaining at time t (SUBROUTINE STATE)
- Expected number of errors remaining at time t and expected number of total errors and imperfect debugging errors detected by time t (SUBROUTINE MEAN)
- MTTF and reliability for given k at time x (SUBROUTINE TBF)

The programs are self-explanatory and include the required subroutines.

TABLE C.1 SUBROUTINE FRSTPT

```
c    subroutine frstpt(n,p,r,n0,et,vt,a,b)--------------------------------
c  function              - compute  mean and variance of first passage time
c                           from n to n0 , and estimate shape and scale
c                           parameters of Gamma distribution
c  usage                 - call frstpt (n,p,r,n0,et,vt,a,b)
c  parameters        n   - (input.) initial no. of errors
c                    p   - (input.) prob. of perfect debugging
c                    r   - (input.) detection rate / error
c                   n0   - (input.) desired no. of errors
c                   et   - (output.) mean time from n to n0
c                   vt   - (output.) variance of time from n to n0
c                    a   - (output.) shape parameter of Gamma distribution
c                    b   - (output.) scale parameter of Gamma distribution
c    -------------------------------------------------------------------
c
     subroutine frstpt (n,p,r,n0,et,vt,a,b)
     x1=0.0
     x2=0.0
     n1=n0+1
     do 55 j=n1,n
     x1=x1+1.0/float(j)
     x2=x2+1.0/float(j)**2
  55 continue
     et=x1/p/r
     vt=x2/p/p/r/r
     b=et/vt
     a=et*b
     return
     end
```

TABLE C.2   SUBROUTINE FPT2

```
c       subroutine fpt2 (n,p,r,n0,t,pdf,cdf)------------------------------------
c  function             - compute p.d.f. and c.d.f. of first passage time from
c                             n to n0
c  usage                - call fpt2 (n,p,r,n0,t,pdf,cdf)
c  parameters         n - (input.) initial no. of errors
c                     p - (input.) prob. of perfect debugging
c                     r - (input.) detection rate / error
c                    n0 - (input.) desired no. of errors
c                     t - (input.) time
c                   pdf - (output.) p.d.f. of first passage time from n to n0
c                   cdf - (output.) c.d.f. of first passage time from n to n0
c  reqd. subroutines    - frstpt, mdgamm
c       -----------------------------------------------------------------------
c
        subroutine fpt2 (n,p,r,n0,t,pdf,cdf)
        call frstpt (n,p,r,n0,et,vt,a,b)
        x=b*t
        x1=(a-1.0)*alog(x)
        x2=algamma(a)
        x3=x1-x-x2
        if (x3 .lt. -88.0) go to 33
        pdf=exp(x3)
        go to 44
     33 pdf=0.0
     44 call mdgamm (x,a,cdf)
        return
     22 cdf=0.0
        pdf=0.0
        return
        end
```

TABLE C.3 SUBROUTINE STATE

```
c     subroutine state (n,p,r,n0,t,pt)-------------------------------------
c  function            - compute probability of being n0 errors remaining
c                            at time t
c  usage               - call state (n,p,r,n0,t,pt)
c  parameters        n - (input.) initial no. of errors
c                    p - (input.) prob. of perfect debugging
c                    r - (input.) detection rate / error
c                   n0 - (input.) desired no. of errors
c                    t - (input.) time
c                   pt - (output.)prob. of being n0 errors at time t
c  read. subroutines   - fpt2
c     ------------------------------------------------------------------
c
      subroutine state (n,p,r,n0,t,pt)
      if (n0.eq.n) go to 33
      call fpt2(n,p,r,n0,t,pdf,cdf)
      pt1=cdf
      go to 44
   33 pt1=1.0
   44 n1=n0-1
      if (n1) 11,22,22
   11 pt2=0.0
      go to 99
   22 call fpt2 (n,p,r,n1,t,pdf,cdf)
      pt2=cdf
   99 pt=pt1-pt2
      return
      end
```

TABLE C.4 SUBROUTINE MEAN

```
c     subroutine mean (n,p,r,t,er,ed0,ed1) -------------------------
c  function          - compute expected no. of errors remaining at time t,
c                      detected, and detected due to imperfect debugging
c                      by time t
c  usage             - call mean (n,p,r,t,er,ed0,ed1)
c  parameters     n  - (input.) initial no. of errors
c                 p  - (input.) prob. of perfect debugging
c                 r  - (input.) detection rate/ error
c                 t  - (input.) time
c                er  - (output.) expected no. of errors remaining at time t
c               ed0  - (output.) expected no. of errors detected by time t
c               ed1  - (output.) expected no. of imperfect debugging errors
c                                detected by time t
c     -----------------------------------------------------------------
c
      subroutine mean (n,p,r,t,er,ed0,ed1)
      er=float(n)*exp(-p*r*t)
      ed0=(float(n)-er)/p
      ed1=ed0*(1.0-p)
      return
      end
```

TABLE C.5 SUBROUTINE TBF

```
c      subroutine tbf (n,p,r,k,x,rel,xmttf)-----------------------------------
c  function              - compute reliability and mttf
c  usage                 - call tbf (n,p,r,k,x,rel,xmttf)
c  parameters          n - (input.) initial no. of errors
c                      p - (input.) prob. of perfect debugging
c                      r - (input.) dec tion rate / error
c                      k - (input.) k-th failure
c                      x - (input.) time
c                    rel - (output.) reliability at time x after (k-1)st failure
c                  xmttf - (output.) mean time between (k-1)st and
c                                    k-th failures
c      ---------------------------------------------------------------------
c
       subroutine tbf (n,p,r,k,x,rel,xmttf)
       xmttf=1.0/(float(n)-p*float(k-1))/r
       rel=exp(-x/xmttf)
       return
       end
```

TABLE C.6   SUBROUTINE MDGAMM

```
c      subroutine mdgamm (x,p,prob)--------------from imsl-----------------
c   function              - compute incomplete gamma distribution
c   usage                 - call mdgamm (x,p,prob)
c   parameters          x - (input.) value to which gamma is to be integrated
c                       p - (input.) gamma parameter
c                    prob - (output.) prob.=integral of gamma(p) to x
c   reqd. subroutines     - gamma
c      -------------------------------------------------------------------
c
       subroutine mdgamm (x,p,prob)
       dimension v(6),v1(6)
       equivalence (v(3),v1(1))
       prob=0.0
       if (x .ge. 0.0) go to 5
       go to 9000
     5 if (p .gt. 0.0) go to 10
       go to 9000
    10 if (x .eq. 0.0) go to 9005
       fnlg=algamma(p)
       cnt=p*alog(x)
       ycnt=x+fnlg
       if ((cnt-ycnt) .gt. -88.0) go to 15
       ax=0.0
       go to 20
    15 ax=exp(cnt-ycnt)
    20 big=1.e35
       cut=1.e-8
       if ((x .le. 1.0) .or. (x .lt. p)) go to 40
       y=1.0-p
       z=x+y+1.0
       cnt=0.0
       v(1)=1.0
       v(2)=x
       v(3)=x+1.0
       v(4)=z*x
       prob=v(3)/v(4)
    25 cnt=cnt+1.0
       y=y+1.0
       z=z+2.0
       ycnt=y*cnt
       v(5)=v1(1)*z-v(1)*ycnt
       v(6)=v1(2)*z-v(2)*ycnt
       if (v(6) .eq. 0.0) go to 50
       ratio=v(5)/v(6)
       reduc=abs(prob-ratio)
       if (reduc .gt. cut) go to 30
       if (reduc .le. ratio*cut) go to 35
```

TABLE C.6 (Continued)

```
  30 prob=ratio
     go to 50
  35 prob=1.0-prob*ax
     go to 9005
  40 ratio=p
     cnt=1.0
     prob=1.0
  45 ratio=ratio+1.0
     cnt=cnt*x/ratio
     prob=prob+cnt
     if (cnt .gt. cut) go to 45
     prob=prob*ax/p
     go to 9005
  50 do 55 i=1,4
     v(i)=v1(i)
  55 continue
     if (abs(v(5)) .lt. big) go to 2
     do 60 i=1,4
     v(i)=v(i)/big
  60 continue
     go to 25
9000 continue
9005 return
     end


     function algamma(p)
     if (p .gt. 31.0) go to 15
     call gamma (p,gp)
     algamma=alog(gp)
     return
  15 z1=(p-0.5)*alog(p)-p+0.5*alog(2.0*3.1415)
     z2=1.0/12.0/p
     z3=1.0/360.0/p/p/p
     z4=1.0/1260.0/p/p/p/p/p
     z5=1.0/1680.0/p/p/p/p/p/p/p
     algamma=z1+z2-z3+z4-z5
     return
     end
```

TABLE C.7   SUBROUTINE GAMMA

```
c       subroutine gamma (xx,gx) -------------------------from imsl-------
c   function            - compute a gamma function of parameter xx
c   usage               - call gamma (xx,gx)
c   parameters       xx - (input.) parameter of gamma function
c                    gx - (output.) value of gamma function
c       --------------------------------------------------------------
c
        subroutine gamma(xx,gx)
        if(xx-57.) 6,6,4
      4 gx=1.0e30
        return
      6 x=xx
        err=1.e-6
        gx=1.
        if(x-2.) 50,50,15
     10 if (x-2.) 110,110,15
     15 x=x-1.
        gx=gx*x
        go to 10
     50 if (x-1.) 60,120,110

     60 if (x-err) 62,62,80
     62 y=float(int(x))-x
        if (abs(y)-err) 120,120,64
     64 if (1.-y-err) 120,120,70

     70 if(x-1.) 80,80,110
     80 gx=gx/x
        x=x+1.
        go to 70
    110 y=x-1.
        gy=1.+y*(-0.5771017+y*(0.9858540+y*(-0.8764218+y*(0.8328212+y*(-0.5684729
\c+y*(0.2548205+y*(-0.0514993)))))))
        gx=gx*gy
    120 return
        end
```

C.2 Programs for Simulation of Imperfect Debugging Data

Computer programs to simulate the data required to estimate the parameters N, p and λ , are given in Tables C.8 to C.11. These programs perform the following functions:

- Simulate data $(\underset{\sim}{t},\underset{\sim}{y})$ for given N, p and λ (SUBROUTINE SMLT)
- Compute the mle's of N, p and λ given the data $(\underset{\sim}{t},\underset{\sim}{y})$ and also obtain the estimate of variance-covariance matrix (SUBROUTINE MLE)
- Compute the Bayesian estimates of N, p and λ for given data $(\underset{\sim}{t},\underset{\sim}{y})$ (SUBROUTINE BAYES)

The programs are self-explanatory.

TABLE C.8 SUBROUTINE SMLT

```
c       subroutine smlt (n,p,r,nn,iseed,t,iy)-----------------------
c  function         - simulate time between s/w failures for
c                     imperfect debugging model
c  usage            - call smlt (n,p,r,nn,iseed,t,iy)
c  parameters     n - (input.)  initial no. of s/w errors
c                 p - (input.)  prob. of perfect debugging
c                 r - (input.)  detection rate / error
c                nn - (input.)  no. of observations for s/w failure time
c             iseed - (input.)  an integer value in the exclusive
c                     range (1,2147483647). iseed is replaced by
c                     a new iseed to be used in subsequent calls.
c                 t - output vector of length nn, containing time
c                     between s/w failures
c                iy - output vector of length nn, indicating the
c                     type of error which is 1 if the i-th
c                     failure is caused by an error due to
c                     imperfect debugging,or 0 otherwith.
c  reqd. subroutine - ggub
c       ------------------------------------------------------------
c
        subroutine smlt (n,p,r,nn,iseed,t,iy)
        dimension t(nn),iy(nn),rr(2)
        nr=n
        ie=0
        do 5 i=1,nn
        if (nr .eq. 0)  go to 99
        xm=1.0/r/float(nr)
        call ggub (iseed,1,rr)
        t(i)=-xm*alog(rr(1))
        call ggub (iseed,1,rr)
        if (rr(1)-(1.0-p)) 22,22,33
     33 nr=nr-1
        go to 44
     22 ie=ie+1
     44 call ggub (iseed,1,rr)
        if (rr(1)-float(ie)/float(nr)) 55,55,66
     66 iy(i)=0
        go to 77
     55 iy(i)=1
        ie=ie-1
     77 print 100,i,t(i),iy(i),nr,ie
    100 format(i5,f15.5,3i5)
      5 continue
     99 return
        end
```

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

TABLE C.9   SUBROUTINE MLE

```
c      subroutine mle (t,iy,nn,en,ep,er,ecov)---------------------------
c   function          - estimate unknown parameters n, p, and lambda
c                       and also estimate variance-covariance
c                       matrix for idm
c   usage             - call mle (t,iy,nn,en,ep,er,ecov)
c   parameters     t  - (input.) a vector of length nn, containing time
c                       between s/w failures
c                 iy  - (input.) a vector of length nn, indicating the
c                       type of error which is 1 if the i-th
c                       failure is caused by an error due to
c                       imperfect debugging, or 0 otherwith
c                  n  - (input.) no. of observations for s/w failure time
c                 en  - (output.) an estimate of parameter n
c                 ep  - (output.) an estimate of parameter p
c                 er  - (output.) an estimate of parameter lambda
c               ecov  - (output.) an estimate of variance-covariance
c                       matrix (3x3)
c      -------------------------------------------------------------
c
      subroutine mle (t,iy,nn,en,ep,er,ecov)
      dimension t(nn),iy(nn),ecov(3,3)
      x1=0.0
      x2=0.0
      y=0.0
      do 5 i=1,nn
      x1=x1+t(i)
      x2=x2+t(i)*float(i-1)
      y=y+float(iy(i))
    5 continue
      en0=float(nn+1)
      ep0=1.0
      do 15 j=1,20
      x3=x1*en0-ep0*x2
      x4=0.0
      x5=0.0
      do 25 i=1,nn
      x4=x4+float(1-iy(i))/(en0-float(i-1))**2
      x5=x5+float(1-iy(i))/(en0-float(i-1))
   25 continue
      f0=x5-x1*float(nn)/x3
      fn=-x4+x1*x1*float(nn)/x3/x3
      fp=-x1*x2*float(nn)/x3/x3
      ph0=float(nn)*(1.0-ep0)*x2/x3-y
      phn=-(1.0-ep0)*float(nn)*x2*x1/x3/x3
      php=float(nn)*x2*(-1.0+(1.0-ep0)*x2/x3)/x3
      hk=fn*php-phn*fp
      hh=-(f0*php-ph0*fp)/hk
      xk=-(fn*ph0-phn*f0)/hk
      en0=en0+hh
      ep0=ep0+xk
```

TABLE C.9 (Continued)

```
      print 100,j,en0,ep0
  100 format(i5,2e15.5)
      if (amax1(abs(hh/en0),abs(xk/ep0)) .le. 0.00001) go to 11
   15 continue
   11 en=en0
      ep=ep0
      er=float(nn)/x3
      rnn=0.0
      rpr=0.0
      rnr=0.0
      do 35 i=1,nn
      rnn=rnn+1.0/(en-float(i-1))/(en-ep*float(i-1))
      rpr=rpr+float(i-1)/(en-ep*float(i-1))
      rnr=rnr+1.0/(en-ep*float(i-1))
   35 continue
      rnr=rnr/er
      rpp=rpr/(1.0-ep)
      rpr=-rpr/er
      rrr=float(nn)/er/er
      rx=rnn*rpp*rrr-rnr*rpp*rnr-rnn*rpr*rpr
      ecov(1,1)=(rpp*rrr-rpr*rpr)/rx
      ecov(1,2)=rpr*rnr/rx
      ecov(1,3)=-rpp*rnr/rx
      ecov(2,2)=(rnn*rrr-rnr*rnr)/rx
      ecov(2,3)=-rnn*rpr/rx
      ecov(3,3)=rnn*rpp/rx
      ecov(2,1)=ecov(1,2)
      ecov(3,1)=ecov(1,3)
      ecov(3,2)=ecov(2,3)
      return
      end
```

TABLE C.10   SUBROUTINE BAYES

```
c      subroutine bayes (t,iy,nn,alpha,beta,pi,rho,xmu,gamma,bn,bp,br)---------
c   function         - obtain bayesian estimates of unknown parameters n, p,
c                          and lambda for idm
c   usage            - call bayes (t,iy,nn,alpha,beta,pi,rho,xmu,gamma,bn,bp,br)
c   parameters    t - (input.) a vector of length nn, containing time
c                          between s/w failures
c                iy - (input.) a vector of length nn, indicating the
c                          type of error which is 1 if the i-th
c                          failure is caused by an error due to
c                          imperfect debugging, or 0 otherwith
c                nn - (input.) no. of observations for s/w failure time
c             alpha - (input.) shape parameter of gamma prior for n
c              beta - (input.) scale parameter of gamma prior for n
c                pi - (input.) first parameter of beta prior for p
c               rho - (input.) second parameter of beta prior for p
c               xmu - (input.) shape parameter of gamma prior for lambda
c             gamma - (input.) scale parameter of gamma prior for lambda
c                bn - (output.) bayesian estimate of n
c                bp - (output.) bayesian estimate of p
c                br - (output.) bayesian estimate of lambda
c      -----------------------------------------------------------------------
c
       subroutine bayes (t,iy,nn,alpha,beta,pi,rho,xmu,gamma,bn,bp,br)
       dimension t(nn),iy(nn)
       x1=0.0
       x2=0.0
       y=0.0
       do 5 i=1,nn
       x1=x1+t(i)
       x2=x2+t(i)*float(i-1)
       y=y+float(iy(i))
     5 continue
       bn0=float(nn+1)
       bp0=1.0
       do 15 j=1,20
       x3=x1*bn0-bp0*x2+gamma
       x4=0.0
       x5=0.0
```

TABLE C.10   (Continued)

```
     do 25 i=1,nn
     x4=x4+float(1-iy(i))/(bn0-float(i-1))**2
     x5=x5+float(1-iy(i))/(bn0-float(i-1))
  25 continue
     f0=x5-x1*(xmu+float(nn-1))/x3+(alpha-1.0)/bn0-beta
     fn=-x4+x1*x1*(xmu+float(nn-1))/x3/x3-(alpha-1.0)/bn0/bn0
     fp=-x4*x2*(xmu+float(nn-1))/x3/x3
     ph0=(xmu+float(nn-1))*(1.0-bp0)*x2/x3-y+(pi-1.0)*(1.0-bp0)/bp0-(rho-1.0)
     phn=-(1.0-bp0)*(xmu+float(nn-1))*x2*x1/x3/x3
     php=(xmu+float(nn-1))*x2*(-1.0+(1.0-bp0)*x2/x3)/x3-(pi-1.0)/bp0/bp0
     hk=fn*php-phn*fp
     hh=-(f0*php-ph0*fp)/hk
     xk=-(fn*ph0-phn*f0)/hk
     bn0=bn0+hh
     bp0=bp0+xk
     print 100,j,bn0,bp0
 100 format(i5,2e15.5)
     if (amax1(abs(hh/bn0),abs(xk/bp0)) .le. 0.00001) go to 11
  15 continue
  11 bn=bn0
     bp=bp0
     br=(xmu+float(nn-1))/x3
     return
     end
```

TABLE C.11   SUBROUTINE GGUB

```
c      subroutine ggub (iseed,n,r)--------------from imsl-----------
c  function      - basic uniform (0,1) pseudo-random number
c                     generator
c  usage         - call ggub (iseed,n,r)
c  parameters iseed - (input.) an integer value in the exclusive
c                     range (1,2147483647). iseed is replaced by
c                     a new iseed to be used in subsequent calls.
c                n - (input.) no. of deviates to be generated
c             r(n) - (output vector of length n,  containing the
c                     deviates in (0,1)
c      -------------------------------------------------------------
c
       subroutine ggub(iseed,n,r)
       dimension r(1)
       double precision z,dpm,dp
       dpm=dble(float(2**31-1))
       dp=dble(float(1/2**31))
       z=iseed
       do 5 i=1,n
       z=dmod(16807.d0*z,dpm)
     5 r(i)=z/dpm
       iseed=z
       return
       end
```

C.3 Programs for the Imperfect Maintenance Model of Section 3

Computer programs to compute the following quantities of interest for given N, p, λ and μ are given in Tables C.12 to C.18.

- Mean and variance of the first passage time from N to n_0 (SUBROUTINE COMP)
- Pdf and cdf of the first passage time from N to n_0 (SUBROUTINE FIRST)
- Probability of the software system being operational with n_0 remaining errors at time t (SUBROUTINE STT)
- Software system availability at time t (SUBROUTINE AVAIL)
- Expected number of errors detected and corrected by time t (SUBROUTINE EXPCT)

The programs are self-explanatory and include the required subroutines (MDGAMM and GAMMA).

TABLE C.12   SUBROUTINE COMP

```
c     subroutine comp (n,p,r,xm,k1,k2,et,vt,a,b,r1,r2)------------------
c  function                  - compute mean and variance of first passage
c                              time, estimate the gamma parameters, and
c                              obtain the constants r1 and r2
c  usage                     - call comp (n,p,r,xm,k1,k2,et,vt,a,b,r1,r2)
c  parameters              n - (input.) initial no. of errors
c                          p - (input.) prob. of perfect debugging
c                          r - (input.) detection rate / error
c                         xm - (input.) correction rate / error
c                         k1 - (input.) first destination
c                         k2 - (input.) second destination
c                         et - (output.) mean first passage time
c                         vt - (output.) variance of first passage time
c                          a - (output.) shape parameter of gamma distribution
c                          b - (output.) scale parameter of gamma distribution
c                         r1 - (output.) smaller constant
c                         r2 - (output.) larger constant
c     ------------------------------------------------------------------
c
      subroutine comp (n,p,r,xm,k1,k2,et,vt,a,b,r1,r2)
      kk1=k1+1
      kk2=k2+1
      rr=sqrt((r+xm)**2-4.0*p*r*xm)
      r1=(r+xm-rr)/2.0
      r2=(r+xm+rr)/2.0
      er1=0.0
      er2=0.0
      vr1=0.0
      vr2=0.0
      if (kk1 .gt. n) go to 11
      do 5 i=kk1,n
      er1=er1+1.0/float(i)
      vr1=vr1+1.0/float(i)/float(i)
    5 continue
   11 er1=er1/r1
      vr1=vr1/r1/r1
      if (kk2 .gt. n) go to 22
      do 15 i=kk2,n
      er2=er2+1.0/float(i)
      vr2=vr2+1.0/float(i)/float(i)
   15 continue
   22 er2=er2/r2
      vr2=vr2/r2/r2
      et=er1+er2
      vt=vr1+vr2
      if (vt .eq. 0.0) go to 33
      b=et/vt
      go to 99
   33 b=0.0
   99 a=et*b
      return
      end.
```

TABLE C.13   SUBROUTINE FIRST

```
c       subroutine first (n,p,r,xm,k1,k2,t,pdf,cdf,r1,r2)-------------
c  function            - compute pdf and cdf of first passage time
c                          and also obtain the constants r1 and r2
c  usage               - call first (n,p,r,xm,k1,k2,t,pdf,cdf,r1,r2)
c  parameters        n - (input.) initial no. of errors
c                    p - (input.) prob. of perfect debugging
c                    r - (input.) detection rate / error
c                   xm - (input.) correction rate / error
c                   k1 - (input.) first destination
c                   k2 - (input.) second destination
c                  pdf - (output.) pdf of first passage time
c                P cdf - (output.) cdf of first passage time
c                   r1 - (output.) smaller constant
c                   r2 - (output.) larger constant
c  reqd. subroutines   - comp, mdgamm, gamma
c       ------------------------------------------------------------
c
        subroutine first (n,p,r,xm,k1,k2,t,pdf,cdf,r1,r2)
        if (min0(k1,k2) .lt. 0) go to 22
        call comp (n,p,r,xm,k1,k2,et,vt,a,b,r1,r2)
        if (b .eq. 0.0) go to 11
        x=b*t
        x1=(a-1.0)*alog(x)
        x2=algamma(a)
        x3=x1-x-x2
        if (x3 .lt. -88.0) go to 33
        pdf=exp(x3)
        go to 44
     33 pdf=0.0
     44 call mdgamm (x,a,cdf)
        return
     11 cdf=1.0
        pdf=1.0
        return
     22 cdf=0.0
        pdf=0.0
        return
        end
```

TABLE C.14   SUBROUTINE STT

```
c     subroutine stt (n,p,r,xm,n0,t,prob)-------------------------
c  function                - compute the probability of being n0 s/w
c                                 errors remaining at time t
c  usage                   - call stt (n,p,r,xm,n0,t,prob)
c  parameters           n - (input.) initial no. of errors
c                       p - (input.) prob. of perfect debugging
c                       r - (input.) detection rate / error
c                      xm - (input.) correction rate / error
c                      n0 - (input.) specified no. of errors
c                       t - (input.) time
c                    prob - (output.) prob. of being n0 errors remaining
c                                 at time t
c  read. subroutine        - first
c     -------------------------------------------------------------
c

      subroutine stt (n,p,r,xm,n0,t,prob)
      call first (n,p,r,xm,n0,n0,t,pdf,cdf,r1,r2)
      prob=cdf
      k=n0-1
      c1=(r-r2)/(r1-r2)
      c2=(r1-r)/(r1-r2)
      call first (n,p,r,xm,k,n0,t,pdf,cdf,r1,r2)
      prob=prob-c1*cdf
      call first (n,p,r,xm,n0,k,t,pdf,cdf,r1,r2)
      prob=prob-c2*cdf
      return
      end
```

TABLE C.15   SUBROUTINE AVAIL

```
c     subroutine avail (n,p,r,xm,t,at)----------------------------
c  function              - compute s/w system availability at time t
c  usage                 - call avail (n,p,r,xm,t,at)
c  parameters          n - (input.) initial no. of errors
c                      p - (input.) prob. of perfect debugging
c                      r - (input.) detection rate / error
c                     xm - (input.) correction rate / error
c                      t - (input.) time
c                     at - (output.) availability at time t
c  read. subroutine      - stt
c     --------------------------------------------------------------
c
      subroutine avail (n,p,r,xm,t,at)
      at=0.0
      n1=n+1
      do 5 i=1,n1
      n0=i-1
      call stt (n,p,r,xm,n0,t,prob)
      at=at+prob
    5 continue
      return
      end
```

TABLE C.16   SUBROUTINE EXPCT

```
c     subroutine expct (n,p,r,xm,t,xmd,xmc)--------------------------
c  function             - compute mean no. of errors detected and
c                            corrected by time t
c  usage                - call expct (n,p,r,xm,t,xmd,xmc)
c  parameters         n - (input.) initial no. of errors
c                     p - (input.) prob. of perfect debugging
c                     r - (input.) detection rate / error
c                    xm - (input.) correction rate / error
c                     t - (input.) time
c                   xmd - (output.) mean no. of errors detected by
c                            time t
c                   xmc - (output.) mean no. of errors corrected by
c                            time t
c  reqd. subroutine     - first
c     -----------------------------------------------------------------
c
      subroutine expct (n,p,r,xm,t,xmd,xmc)
      h1=0.0
      h2=0.0
      h=0.0
      do 5 i=1,n
      k=i-1
      call first (n,p,r,xm,k,i,t,pdf,cdf,r1,r2)
      h1=h1+cdf
      call first (n,p,r,xm,i,k,t,pdf,cdf,r1,r2)
      h2=n2+cdf
      call first (n,p,r,xm,k,k,t,pdf,cdf,r1,r2)
      h=h+cdf
    5 continue
      xmc=h/p
      xmd=(h1*(1.0-xm/r1)+h2*(xm/r2-1.0))*r/(r1-r2)
      return
      end
```

TABLE C.17   SUBROUTINE MDGAMM

```
c      subroutine mdgamm (x,p,prob)--------------from imsl-----------------
c   function           - compute incomplete gamma distribution
c   usage              - call mdgamm (x,p,prob)
c   parameters       x - (input.) value to which gamma is to be integrated
c                    p - (input.) gamma parameter
c                 prob - (output.) prob.=integral of gamma(p) to x
c   reqd. subroutines  - gamma
c      ---------------------------------------------------------------------
c
       subroutine mdgamm (x,p,prob)
       dimension v(6),v1(6)
       equivalence (v(3),v1(1))
       prob=0.0
       if (x .ge. 0.0) go to 5
       go to 9000
     5 if (p .gt. 0.0) go to 10
       go to 9000
    10 if (x .eq. 0.0) go to 9005
       fnlg=algamma(p)
       cnt=p*alog(x)
       ycnt=x+fnlg
       if ((cnt-ycnt) .gt. -88.0) go to 15
       ax=0.0
       go to 20
    15 ax=exp(cnt-ycnt)
    20 big=1.e35
       cut=1.e-8
       if ((x .le. 1.0) .or. (x .lt. p)) go to 40
       y=1.0-p
       z=x+y+1.0
       cnt=0.0
       v(1)=1.0
       v(2)=x
       v(3)=x+1.0
       v(4)=z*x
       prob=v(3)/v(4)
    25 cnt=cnt+1.0
       y=y+1.0
       z=z+2.0
       ycnt=y*cnt
       v(5)=v1(1)*z-v(1)*ycnt
       v(6)=v1(2)*z-v(2)*ycnt
       if (v(6) .eq. 0.0) go to 50
```

TABLE C.17   (Continued)

```
      ratio=v(5)/v(6)
      reduc=abs(prob-ratio)
      if (reduc .gt. cut) go to 30
      if (reduc .le. ratio*cut) go to 35
   30 prob=ratio
      go to 50
   35 prob=1.0-prob*ax
      go to 9005
   40 ratio=p
      cnt=1.0
      prob=1.0
   45 ratio=ratio+1.0
      cnt=cnt*x/ratio
      prob=prob+cnt
      if (cnt .gt. cut) go to 45
      prob=prob*ax/p
      go to 9005
   50 do 55 i=1,4
      v(i)=v1(i)
   55 continue
      if (abs(v(5)) .lt. big) go to 25
      do 60 i=1,4
      v(i)=v(i)/big
   60 continue
      go to 25
 9000 continue
 9005 return
      end


      function algamma(p)
      if (p .gt. 31.0) go to 15
      call gamma (p,gp)
      algamma=alog(gp)
      return
   15 z1=(p-0.5)*alog(p)-p+0.5*alog(2.0*3.1415)
      z2=1.0/12.0/p
      z3=1.0/360.0/p/p/p
      z4=1.0/1260.0/p/p/p/p/p
      z5=1.0/1680.0/p/p/p/p/p/p/p
      algamma=z1+z2-z3+z4-z5
      return
      end
```

TABLE C.18 *SUBROUTINE GAMMA*

```
c     subroutine gamma (xx,gx) --------------------------from imsl-------
c  function            - compute a gamma function of parameter xx
c  usage               - call gamma (xx,gx)
c  parameters       xx - (input.) parameter of gamma function
c                   gx - (output.) value of gamma function
c     ---------------------------------------------------------------
c
      subroutine gamma(xx,gx)
      if(xx-57.) 6,6,4
    4 gx=1.0e30
      return
    6 x=xx
      err=1.e-6
      gx=1.
      if(x-2.) 50,50,15
   10 if (x-2.) 110,110,15
   15 x=x-1.
      gx=gx*x
      go to 10
   50 if (x-1.) 60,120,110

   60 if (x-err) 62,62,80
   62 y=float(int(x))-x
      if (abs(y)-err) 120,120,64
   64 if (1.-y-err) 120,120,70

   70 if(x-1.) 80,80,110
   80 gx=gx/x
      x=x+1.
      go to 70
  110 y=x-1.
      gy=1.+y*(-0.5771017+y*(0.9858540+y*(-0.8764218+y*(0.8328212+y*(-0.5684729
\c+y*(0.2548205+y*(-0.0514993)))))))
      gx=gx*gy
  120 return
      end
```

C.4 Program for Bayesian Software Correction Limit Policies (Section 5)

Computer programs to compute the optimum policies for model 1 and model 2 are given in Tables C.19 to C.23. These programs perform the following functions:

- For model 1, simulate error occurrence time and correction time and then compute Bayesian estimates of mean correction time, optimum correction limit time, and its minimum cost per unit time (SUBROUTINE MDL1)
- For model 2, simulate error occurrence time and correction time and then compute Bayesian estimate of mean correction time, optimum ocrrection limit time, and its minimum cost per unit times, and also provide the optimum sample size (SUBROUTINE MDL2)

The programs are self-explanatory and include the required subroutines (eg. DATA 1, GGUB and OPTMM).

TABLE C.19   SUBROUTINE MDL1

```
c      subroutine mdl1 (iseed,nn,r,xmu1,xmu2,c,alpha,beta,xn,yn,tt,ct)--------
c   function          - simulate error occurrence time and correction time
c                         and then compute bayesian estimate of mean
c                         correction time, optimum correction limit time,
c                         and its minimum cost per unit time
c   usage             - call mdl1 (iseed,nn,r,xmu1,xmu2,c,alpha,beta,xn,yn,tt,ct)
c   parameters iseed - (input.) an integer value in the exclusive range
c                        (1,2147483647). iseed is replaced by a new iseed
c                        to be used in subsequent calls.
c                  nn - (input.) no. of observations
c                   r - (input.) error occurrence rate
c                xmu1 - (input.) mean correction time in phase 1
c                xmu2 - (input.) mean correction time in phase 2
c                   c - (input.) a vector of length 3
c                        c(1) - cost per unit time of error correction
c                               in pase 1
c                        c(2) - cost per unit time of error correction
c                               in pase 2
c                        c(3) - sampling cost per sample size
c               alpha - (input.)  a vector of length 2
c                        alpha(1) - shape parameter of inverted gamma
c                                   prior for mean correction time
c                                   in pase 1
c                        alpha(2) - shape parameter of inverted gamma
c                                   prior for error occurrence rate
c                beta - (input.) a vector of length 2
c                        beta(1) - scale parameter of inverted gamma
c                                  prior for mean correction time
c                                  in pase 1
c                        beta(2) - scale parameter of inverted gamma
c                                  prior for error occurrence rate
c                  xn - (output.) bayesian estimate of error occurence rate
c                  yn - (output.) bayesian estimate of mean correction
c                        time in pase 1
c                  tt - (output.) optimum correction limit time
c                  ct - (output.) minimum cost per unit time
c   reqd. subroutines  -  data1, optmm
c      -----------------------------------------------------------------------
c
      subroutine mdl1 (iseed,nn,r,xmu1,xmu2,c,alpha,beta,xn,yn,tt,ct)
      dimension c(3),alpha(2),beta(2),xy(2)
      call data1 (iseed,nn,r,xmu1,xx,yy)
      xn=(xx+beta(2))/(alpha(2)+float(nn-1))
      a=0.0
      b=xn
      call optmm (nn,yy,c,alpha,beta,xmu2,a,b,yn,tt,ct)
      return
      end
```

AD-A067 690   SYRACUSE UNIV N Y   F/G 9/2
BAYESIAN SOFTWARE PREDICTION MODELS. VOLUME V. SUMMARY OF TECHN--ETC(U)
OCT 78 A L GOEL   F30602-76-C-0097
UNCLASSIFIED   TR-78-5   RADC-TR-78-155-VOL-5   NL

2 OF 2
ADA
067690

END
DATE
FILMED
6 79
DDC

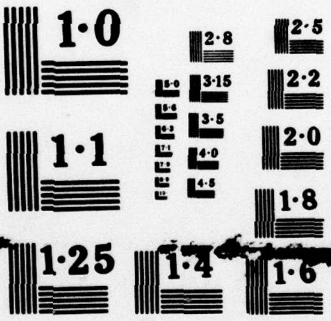

1·0
2·8
2·5
3·15
2·2
3·5
2·0
1·1
4·0
4·5
1·8
1·25
1·4
1·6

TABLE C.20   SUBROUTINE MDL2

```
c    subroutine mdl2 (iseed,r,xmu1,xmu2,c,alpha,beta,nn,xn,yn,tt,ct)--------
c  function          - simulate error occurrence time and correction time
c                      and then compute bayesian estimate of mean
c                      correction time, optimum correction limit time,
c                      and its minimum cost per unit time, and also
c                      provide optimum sample size
c  usage             - call mdl2 (iseed,r,xmu1,xmu2,c,alpha,beta,nn,xn,yn,tt,ct)
c  parameters iseed  - (input.) an integer value in the exclusive range
c                      (1,2147483647).  iseed is replaced by a new iseed
c                      to be used in subsequent calls.
c                r   - (input.) error occurrence rate
c             xmu1   - (input.) mean correction time in phase 1
c             xmu2   - (input.) mean correction time in phase 2
c                c   - (input.) a vector of length 3
c                      c(1) - cost per unit time of error correction
c                             in pase 1
c                      c(2) - cost per unit time of error correction
c                             in pase 2
c                      c(3) - sampling cost per sample size
c            alpha   - (input.)  a vector of length 2
c                      alpha(1) - shape parameter of inverted gamma
c                                 prior for mean correction time
c                                 in pase 1
c                      alpha(2) - shape parameter of inverted gamma
c                                 prior for error occurrence rate
c             beta   - (input.) a vector of length 2
c                      beta(1) - scale parameter of inverted gamma
c                                prior for mean correction time
c                                in pase 1
c                      beta(2) - scale parameter of inverted gamma
c                                prior for error occurrence rate
c               nn   - (output.) optimum sample size
c               xn   - (output.) bayesian estimate of ereor occurence rate
c               yn   - (output.) bayesian estimate of mean correction
c                      time in pase 1
c               tt   - (output.) optimum correction limit time
c               ct   - (output.) minimum cost per unit time .
c  reqd. subroutines  -  gsub, optmm
c    -----------------------------------------------------------------------
c
     subroutine mdl2 (iseed,r,xmu1,xmu2,c,alpha,beta,nn,xn,yn,tt,ct)
```

TABLE C.20 (Continued)

```
      dimension c(3),alpha(2),beta(2),xy(2)
      xx=0.0
      yy=0.0
      do 5 nn=1,100
      call ssub (iseed,2,xy)
      xx=xx-r*alog(xy(1))
      yy=yy-xmu1*alog(xy(2))
      if (nn .eq. 1) go to 5
      xn=(xx+beta(2))/(alpha(2)+float(nn-1))
      a=c(3)*float(nn)
      b=xx+xn+yy
      call optmm (nn,yy,c,alpha,beta,xmu2,a,b,yn,t1,c1)
      a=c(3)*float(nn+1)
      b=xx+2.0*xn+yy+yn
      call optmm (nn,yy,c,alpha,beta,xmu2,a,b,yn,t2,c2)
      print 100,nn,xn,yn,t1,c1,t2,c2
  100 format(i5,6e15.5)
      if (t1-t2) 11,22,5
   22 if (c1-c2) 11,11,5
    5 continue
   11 tt=t1
      ct=c1
      return
      end
```

TABLE C.21   SUBROUTINE DATA1

```
c     subroutine data1 (iseed,nn,r,xmu1,xx,yy)--------------------------
c  function              - simulate error occurrence time and correction
c                          time in phase 1
c  usage                 - call data1 (iseed,nn,r,xmu1,xx,yy)
c  parameters    iseed - (input.) an integer value in the exclusive
c                          range (1,2147483647).  iseed is replaced
c                          by a new iseed to be used in subsequent calls.
c                   nn - (input.) sample size
c                    r - (input.) error occurrence rate
c                 xmu1 - (input.) mean correction time in phase 1
c                   xx - (output.) total amount of error occurrence time
c                   yy - (output.) total amount of error correction time
c                          in phase 1
c  reqd. subroutine    - ssub
c     ------------------------------------------------------------------
c
      subroutine data1 (iseed,nn,r,xmu1,xx,yy)
      dimension xy(2)
      xx=0.0
      yy=0.0
      do 5 i=1,nn
      call ssub (iseed,2,xy)
      xx=xx-r*alog(xy(1))
      yy=yy-xmu1*alog(xy(2))
    5 continue
      return
      end
```

TABLE C.22 SUBROUTINE GGUB

```
c      subroutine ggub (iseed,n,r)---------------from imsl-----------
c  function         - basic uniform (0,1) pseudo-random number
c                       generator
c  usage            - call ggub (iseed,n,r)
c  parameters iseed - (input.) an integer value in the exclusive
c                       range (1,2147483647). iseed is replaced by
c                       a new iseed to be used in subsequent calls.
c                 n - (input.) no. of deviates to be generated
c                 r - output vector of length n,  containing the
c                       deviates in (0,1)
c      ---------------------------------------------------------------
c
       subroutine ggub(iseed,n,r)
       dimension r(1)
       double precision z,dpm,dpn,dp
       data dpm,dpn/2147483647.d0,1.d0/
       dp=dpn/(dpm+dpn)
       z=iseed
       do 5 i=1,n
       z=dmod(16807.d0*z,dpm)
     5 r(i)=z*dp
       iseed=z
       return
       end
```

TABLE C.23   SUBROUTINE OPTMM

```
c     subroutine optmm (nn,yy,c,alpha,beta,xmu2,a,b,yn,tt,ct)-----------
c   function            - compute bayesian estimate of mean correction
c                           time, optimum correction limit time, and
c                           its minimum cost per unit time
c   usage               - call optmm (nn,yy,c,alpha,beta,xmu2,a,b,yn,tt,ct)
c   parameters       nn - (input.) no. of observations
c                    yy - (input.) total amount of observed correction
c                           time
c                     c - (input.) a vector of length 3
c                           c(1) - cost per unit time of error correction
c                                    in pase 1
c                           c(2) - cost per unit time of error correction
c                                    in pase 2
c                           c(3) - sampling cost per sample size
c                 alpha - (input.) a vector of length 2
c                           alpha(1) - shape parameter of inverted gamma
c                                        prior for mean correction time
c                                        in pase 1
c                           alpha(2) - shape parameter of inverted gamma
c                                        prior for error occurrence rate
c                  beta - (input.) a vector of length 2
c                           beta(1) - scale parameter of inverted gamma
c                                       prior for mean correction time
c                                       in pase 1
c                           beta(2) - scale parameter of inverted gamma
c                                       prior for error occurrence rate
c                  xmu2 - (input.) mean correction time in pase 2
c                     a - (input.) constant of a in equation (a-1)
c                     b - (input.) constant of b in equation (a-1)
c                    yn - (output.) bayesian estimate of mean correction
c                           time in pase 1
c                    tt - (output.) optimum correction limit time
c                    ct - (output.) minimum cost per unit time
c     -----------------------------------------------------------------
c
      subroutine optmm (nn,yy,c,alpha,beta,xmu2,a,b,yn,tt,ct)
      dimension c(3),alpha(2),beta(2)
      aa=c(2)*b-a
      bb=(c(1)*b-a)/xmu2
      g1=yy+beta(1)
      a0=alpha(1)+float(nn-1)
      yn=g1/a0
      g2=bb/(aa+yn*c(2)-c(1))
      g3=(a0+1.0)/g1
      t=(g3/g2-1.0)*g1
      if (t .lt. 0.0) go to 22
      do  5 i=1,20
      rt=g3/(1.0+t/g1)
      gst=g1/a0*(1.0-(1.0+t/g1)**(-a0))
      gt=(1.0+t/g1)**(-a0-1.0)
      x1=aa+(c(2)-c(1))*gst
      f0=rt*x1+(c(2)-c(1))*gt-bb
      rrt=-g3/g1/(1.0+t/g1)**2
      ff=rrt*x1
      h=-f0/ff
      t=t+h
      print 100,i,t
  100 format(i5,e15.5)
```

TABLE C.23 (Continued)

```
      if (abs(h/t) .le. 0.00001) go to 11
    5 continue
   11 tt=t
      ct=(c(1)-c(2)*xmu2*rt)/(1.0-xmu2*rt)
      return
   22 tt=0.0
      ct=(a+c(2)*xmu2)/(b+xmu2)
      return
      end
```